

JavaTM magazine

By and for the Java community



TAME THE CLOUD

How Java EE is evolving to become the
cloud computing platform standard

10

JAVA VS. SPAM

Mollom eliminates
unwanted postings
from Websites
worldwide

33

PROJECT COIN IS A TIME-SAVER

How Java SE 7's
language changes
streamline your
code

68

FIX THIS

Test your
knowledge
with a Swing
conundrum

//from the editor /



Sometimes, buzzwords are for real. *Cloud computing* is one such example: most will agree that the term itself is overused and lacks a clear, universally accepted definition. But we can also agree on the importance of understanding cloud computing's likely impact on the application development process. Pretending otherwise would constitute a serious breach of career-management duty.

In some quarters, that impact is expressed in the context of the emerging “DevOps” principle, which calls for developers to use their programming powers to free themselves from the burden of IT operations. Cloud computing, by almost any definition, is a key DevOps enabler.

Currently, the Java EE platform lacks the abstractions that allow Java developers to move confidently in that direction. But with [Java EE 7](#) (JSR-342; final release expected later in 2012), that will change—with the inclusion of updated APIs, roles, and more that will bring support for cloud service requirements such as multi-tenancy, elasticity, and scalability directly into the standard. As a result, apps written to target Java EE 7 app servers (which will in fact become services themselves) will be much more natively cloud-aware than they are today. What could be more DevOps-friendly than that?

Our [interview](#) with Cameron Purdy, Oracle vice president of Java EE development, reveals the reasoning behind these efforts in more detail. (You can also [review the Java EE 7 JSRs](#).) And as usual, you'll find this issue packed with informative, actionable articles on other subjects as well—including [wrap-ups](#) of JavaOne Latin America and Devoxx 2011, a [Swing JLayer primer](#) by Josh Marinacci, an [introduction to Java SE 7's Project Coin features](#) by Julien Ponge, and more. Enjoy it!

Justin Kestelyn, Editor in Chief



PHOTOGRAPH BY BOB ADLER



GIVE BACK! ADOPT A JSR

[Find your JSR here](#)



//send us your feedback /

We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.



We didn't invent the Internet...

...but our components help **you** power the apps that bring it to business.



PURE JAVA COMPONENTS & ENTERPRISE ADAPTERS FOR

- **E-Business**
AS2, EDI/X12, NAESB, OFTP ...
- **Credit Card Processing**
Authorize.Net, TSYS, FDMS ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX ...
- **Internet Business**
Amazon, eBay, PayPal ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Secure Email**
S/MIME, OpenPGP ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression & Encryption**
Zip, Gzip, Jar, AES ...

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

NEWS ROUNDUP

JavaOne LATIN AMERICA 2011



Georges Saab

JavaOne Latin America 2011 was held in São Paulo, Brazil, December 6–8, 2011. The regional JavaOne conference included great keynotes, deep technical sessions, engaging hands-on labs, a bustling Oracle Technology Network lounge, booth swag (Duke T-shirts!), wonderful conversations, and even a maté tasting.

PHOTOGRAPHY BY LOAINE GRONER AND ARUN GUPTA



Nandini Ramani

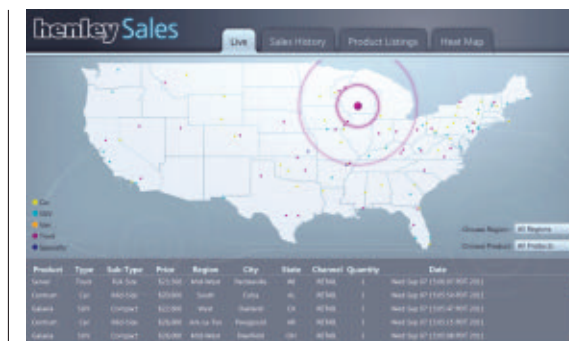
PROMISES DELIVERED

At the Java strategy keynote on the first day of JavaOne Latin America, Oracle executives stepped through Oracle's Java technology roadmap for Java SE, Java EE, Java ME, and JavaFX. "Last year at JavaOne Latin America, Oracle was full of promises," said **Georges Saab**, vice president of development, Java Platform Group. "This year we can say Oracle delivered on those promises: Java SE 7, JavaFX 2.0, and increasing transparency in the JCP [Java Community Process]." **Nandini Ramani**, vice president of development, Java client platform, announced the release of JavaFX 2.0.2. She explained that starting with JavaFX 2.0.2, JavaFX is available under the same license and business model as Java SE. This includes the ability to distribute the JavaFX runtime (or Software Development Kit) for third-party developers with their application(s), subject to the terms and conditions of the license. (Developers can follow and join the JavaFX open source project at OpenJFX.) The technical keynotes on the second day provided more details on each of these technologies.

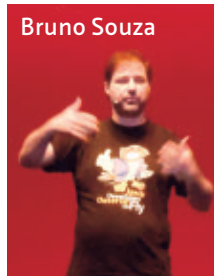


FOCUS ON JAVA FX

There was lots of buzz around JavaFX. At the technical keynote, the JavaFX demos generated the most interest, especially the JavaFX game running on a desktop and two different tablets. The other demo that sparked a lot of interest was the Henley Sales Dashboard, a client/server application for a fictional global automobile company. The front end was developed with JavaFX, displaying real-time sales data via MySQL and deployed on Oracle GlassFish Server. At the standing-room-only session "XML-Free Programming: Java Server and Client," **Stephen Chin**, chief agile methodologist at GXS and Java Champion, and **Arun Gupta**, Java evangelist at Oracle, discussed how to build Web apps with a JavaFX client and a Java EE server.



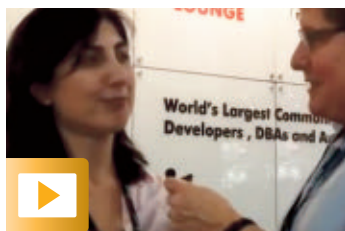
The source code for the Henley Sales Dashboard is available as part of the [JavaFX Sample Showcase](#) (currently only for Windows).



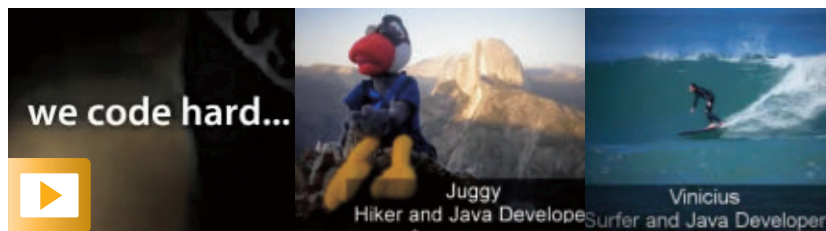
Bruno Souza

BEST FOR LAST: THE COMMUNITY KEYNOTE

The [JavaOne Community Keynote](#) was informative and funny. **Fabiane Nardon** and **Bruno Souza** of Brazilian Java user group SouJava opened the keynote with their list of the top five developments in Java for the year: 5) Java on Mac to OpenJDK, 4) SouJava and the London Java Community getting seats on the JCP Executive Committee, 3) Twitter adopting Java for improved performance, 2) Java SE 7 being released, and 1) JavaFX being made open source. Nardon then discussed women in technology and, with SouJava members **Yara Senger**, **Ana Abrantes**, and **Loiane Groner**, announced the formation of [jDuchessBR](#), a group for Brazilian women using Java. Then **Vinicius Senger** gave a demo of the Duke's Choice Award winner, jHome, showing that what you can do with Java is limited only by your imagination: adjust the lights in your house via Twitter, brew coffee, or monitor someone's heart rate remotely (Arun Gupta was the guinea pig). The keynote concluded with a twist on the [Java Life](#) video entitled [Real Java Geeks](#). See what Java developers do when they aren't in their cubicles (or offices, home offices, or coffee shops).



Fabiane Nardon talks about zero downtime and the amazing performance of [Java EE](#).



When they're not coding hard, Java developers surf, hike, ride motorcycles, fish, practice jujitsu, and so much more.



Geek Bike Ride

THE COMMUNITY THAT RIDES TOGETHER...

In true community spirit, **Fabiane Nardon** and other SouJava members put together a Java community bike ride the Sunday morning before JavaOne Latin America. Nardon arranged for bike rentals and matching jerseys, and "bike angels" made sure no one was left behind. Nearly 30 riders enjoyed slightly overcast and cool weather and a wonderful route through São Paulo, thanks to Sunday lane closures just for bicyclists. The riders looked great in their fabulous Duke bike jerseys. The route was 12 miles (22 kilometers)—check out the [GPS documentation](#) and [full redundancy](#). The group rode at a leisurely pace that allowed everyone to chat and meet new friends. The group even visited two parks and a street fair along the way. Affectionately called the Geek Bike Ride, it had just the right combination of exercise, technology, and fun. Watch for [#geekbikeride](#) to join or start a community bike ride near you.

PHOTOGRAPHY BY ARUN GUPTA
AND TORI WIELDT

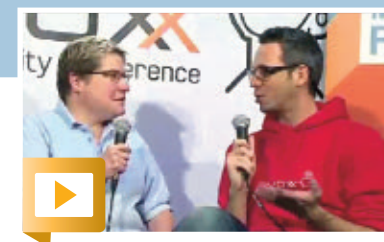
Parleys.com Brings Sessions to You

Couldn't make it to JavaOne or Devovx?

That needn't stop you from "attending" many of the most important conference sessions. You can do so at Parleys, the [GlassFish-powered](#) e-learning platform invented by Java Champion, Belgian Java User Group leader, and Devovx Founder [Stephan Janssen](#). The Parleys team recorded 40 percent of the JavaOne 2011 sessions, along with a substantial sampling of Devovx 2011 sessions and interviews.

A Parleys technical session presentation provides you with a full educational experience, including audio, slides, and video of demos from the sessions. You hear and see almost everything the session's live attendees heard and saw—with the added benefit that you can pause, back up, and review a session segment or slide any time you like. You can watch Parleys presentations online, or you can download them for offline viewing using the free Parleys Desktop application.

Visit the Parleys [JavaOne channel](#) to view popular sessions from JavaOne 2011 including "Introduction to JavaFX 2.0," "Java EE 6: The Cool Parts," and "The Heads and Tails of Project Coin."



Stephan Janssen talks about the challenges of putting on Devovx.



Devoxx 2011 celebrated its 10-year anniversary in Antwerp, Belgium, in November 2011. Sold out every year weeks in advance, Devoxx has become the biggest Java developer conference in Europe, with 3,350 attendees from 40 countries and more than 170 well-known speakers giving 150 sessions over five days.

Henrik Stahl and **Cameron Purdy** of Oracle talked about Oracle's investment in and innovation around the Java platform with the upcoming releases of Java SE 8, Java EE 7, and JavaFX 3.0. **Mark Reinhold**, **Joe Darcy**, and **Brian Goetz**, Java SE architects and technical leads at Oracle, explained the latest progress with [Project Coin](#), [Project Jigsaw](#) (modularity), and [Project Lambda](#) in a series of sessions and casual discussions with developers. Java EE 7 development will be multitenant, on demand, and autoprovisioned for the cloud platform. **Jerome Dochez**, a GlassFish architect at Oracle, discussed cloud infrastructure and



JavaFX 2.0, released at JavaOne, had a

1. Parents should encourage their daughters to play with Legos and learn programming.
2. More organizations should target young women in high school and college to expose them to programming. Women need mentors (men or women) to learn to become speakers at conferences and to promote themselves better.

To celebrate its 10-year success, Devoxx will expand to France, in the first conference modeled after Devoxx in Belgium. At Devoxx France, April 18–20, 2012, 75 percent of the sessions will be in French and 25 percent in English. The Paris Java user group behind this event promises a Devoxx experience plus spring in Paris and, of course, wine. A [call for papers](#) is open, and Oracle is a sponsor.



It's all happening in Paris.

PHOTOGRAPHS COURTESY OF DEVOXX



JAVA TECH

ABOUT US

0

f

ava
net

log



08

A large, modern, red and white structure shaped like a computer monitor, displaying 'ORACLE' and '100 of the 100 Top Telecoms'. The structure is situated on a city street, with people walking past it. A banner for 'Cisco' is visible in the background.

ILLUSTRATION BY I-HUA CHEN, PHOTOGRAPH BY RON SELLERS



JAVA TECH

ABOUT US

○

f

ava
net

olog



09

09

09



09

09

09



09

09

09



09

09

09

A man with short brown hair, glasses, and a goatee is shown in profile, looking out of a large circular window. He is wearing a dark brown jacket over a blue collared shirt and dark blue jeans. His right hand is in his pocket. The window has white horizontal blinds that are partially open, allowing bright light to enter. The window frame is white and circular. The background outside the window is a bright, overexposed outdoor scene.

BY DAVID BAUM AND ED BAUM

Got spam? Who doesn't! But if you think finding and deleting a few hundred spam messages from your inbox or Website is a challenge, try deleting a few hundred million. That's the volume that leading spam-fighters such as Dries Buytaert battle each day. As cocreator of Mollom, a Java-based technology that processes hundreds of requests per second to keep the invading horde at bay, Buytaert is continually confronted with the "dark side" of the internet: it has become a petri dish of sorts,

breeding countless unsolicited automated messages and comments like invasive bacteria. This infestation weakens worldwide productivity and threatens to compromise the way the Web works. The battle for the internet is raging, and it's us against the spambots.

These *spambots*—automated programs that assist in sending and posting spam—are getting smarter and more intrusive all the time. Thankfully, defensive strategies keep getting better too, as evidenced by Buytaert and his team. They recently migrated Mollom from home-

**Dries Buytaert,
Mollom cocreator, at the
company's Burlington,
Massachusetts, office**

PHOTOGRAPHY BY DAVE BRADLEY

Java EE 6



ACCURATE AIM
Mollom's efficiency rate at identifying spam is 99.96 percent. Only 4 messages out of 10,000 succeed in fooling its Java programs.

Web service that keeps invading hordes of spambots at bay, was based on a custom Java SE–based XML-RPC server. To allow increased focus on Mollom’s core Machine Learning technology, Mollom co-creators Dries Buytaert and Benjamin Schrauwen decided to switch to a Java EE 6 back end, hosted on a GlassFish Server Open Source Edition 3.1.1 cluster. The Mollom team now extensively uses several core technologies such as [JAX-RS with Jersey](#), the [Java Persistence API](#), and [Enterprise JavaBeans 3.1](#) (both singleton and stateless session beans). This has enabled the team to quickly develop an extended REST version of their API. For storage, Buytaert and Schrauwen chose a [Cassandra](#) cluster, because the back end is very write-intensive. Cassandra also allows the Mollom team to easily scale the storage layer across multiple data centers. Currently Mollom is running on dedicated machines, and to further reduce maintenance work, Buytaert and Schrauwen plan to move the infrastructure to [Amazon’s cloud](#) environment, and to take advantage of the upcoming cloud features in GlassFish Server Open Source Edition.

Buytaert says. "We know that's not human behavior. It helps us identify and block these malicious automated programs."

By intelligently combining all these techniques and only using the CAPTCHA as a final test to weed out the small number of false positives and false negatives, Mollom provides the best-possible quality in terms of blocking spam while delivering the best-possible user experience for visitors to the sites it protects.

Mollom is a real-time service, eliminating spam, filtering profanity, and safeguarding all kinds of sites—from small blogs that may only get one spam message per week to large social networks such as Europe's Netlog, with its 85 million registered users, and large media companies such as Sony Music Entertainment, which host hundreds of popular artists' sites. When someone makes a comment on a protected site, that site sends the comment to Mollom. Mollom analyzes the content and then returns a reply to the site. "It's important for us to have a low latency so we can process complex statistical analysis in milliseconds," Buytaert explains. "Visitors to Websites don't want to wait after clicking Submit to see if their comments were accepted. To accomplish

Buytaert clears his head by Mollom's "campfire," a place where staffers can brainstorm or just relax.

this, we need a technology that can keep a lot of data in persistent memory so we can instantly classify content."

When Buytaert created Mollom with his partner, Benjamin Schrauwen, they initially built everything from scratch. But as Mollom grew to the point where it was handling millions of messages per day, Buytaert and Schrauwen found that they were spending too much time on infrastructure-related issues. That's when they switched to GlassFish Server Open Source Edition.

"We migrated to GlassFish because it let us worry less about memory management, XML parsing, database connection pooling, REST handling, and a lot of other factors that make big information systems work," Buytaert says. "It lets us scale that runtime environment more effectively. By migrating from our home-grown Java-based solution to GlassFish, we freed ourselves to focus more on the domain-specific challenges of Mollom. It's a great fit."

Buytaert says new users can get Mollom up and running within minutes. They simply create an account on mollom.com, install a plug-in, and enter public and private keys.

"The volume of spam comments we block is one measure of Mollom's success, but the



quality of our service is equally important," Buytaert says. "Our efficiency rate is currently 99.96 percent. That means if we see 10,000 messages, we make four mistakes. We know we are helping to keep the Web a little bit cleaner. In business terms, that means Website owners can spend a lot less time worrying about the content that doesn't belong and more time creating valuable services for their user base. [</article>](#)

David Baum and **Ed Baum** are freelance writers specializing in innovative businesses, emerging technologies, and compelling lifestyles.



SNAPSHOT

**CHOICE HOTELS
INTERNATIONAL**

choicehotels.com

Headquarters:

Silver Spring, Maryland

Industry:

Hospitality

Revenue:

US\$642 million in FY 2010

Employees:

1,800 (U.S. only)

Java version used:

Java Development Kit 6

That's why Choice Hotels, one of the world's largest lodging chains, was intrigued by the prospect of adding Google to its portfolio of distribution partners. The Web search giant asked Choice to participate in a new information service that it was developing for its popular Google Maps and Google Places services. The new service, launched in July 2011, displays real-time hotel room availability and rates alongside location maps.

"If you go to maps.google.com or even just google.com and search for hotels in Phoenix or Los Angeles, for instance, you'll get a map with pinpoints with letters—A, B, C, D—next to them," explains Fletcher. "The letters correspond to a hotel that's on the nav [navigation] bar at left. The nav bar displays information about that hotel, including the name, location, and star rating. Then there's a little

drop-down box that shows you the hotel and the rate for that hotel from a number of different sources. If you click on the link for Choice Hotels, it takes you to our Website to book that specific hotel.”

That link is a “deep” click-through that brings potential customers directly to Choice’s own Website—where, it is hoped, they will book a room. According to Fletcher, Java was uniquely suited as the language and platform to create this unique, innovative service.

"Java is a true enterprise-class development platform," says Fletcher. "We operate at internet scale. We do US\$7 billion in revenue a year through our central channels,

so we definitely have enterprise-class problems to solve. Java is uniquely positioned as a development platform to solve those.”

THE CORE OF CHOICE

Founded in 1939 and based in Silver Spring, Maryland, with IT operations in Phoenix, Arizona, Choice Hotels is one of the lead-

ing lodging companies in the world. Structured on the franchise model, the chain has more than 6,000 hotels in the U.S. and in more than 30 countries around the globe. What began as a chain of motels has blossomed into Choice's 11 brands, which range from budget Econo Lodge and Rodeway Inns and

EARLY DAYS

Choice started out as a loose affiliation of seven independent motel owners in Florida in the 1930s.



Fletcher catches up with his staff during a hallway mini-meeting; a Choice staffer crushes his ping-pong opponent (an Oracle rep) while Fletcher looks on.

midrange Quality Inns and Comfort Suites to full-service Cambria Suites and the boutique and historical hotels that are part of the Ascend Collection.

Like others in the lodging industry, Choice's IT infrastructure is a hodgepodge of decades-old legacy systems, some based on the first computerized reservation system—originally designed for the airline industry 25 years ago—mixed in with newer systems based on proprietary languages and protocols and some more-recent technologies, such as C, Java, and even a little .NET. Its core is its central reservation system (CRS), which stores up-to-date information about Choice's inventory of rooms, room rates, and availability. Layered on top are applications and processes that push

that information out to its Website, reservation call center, and third-party distributors.

Twenty years ago, these distributors included global distribution systems (GDSs), such as Sabre, Worldspan, and Galileo, and members of the Open Travel Alliance (OTA). When the travel industry began to establish a presence online, Choice added online travel bureaus such as Expedia, Travelocity, and Orbitz to the mix as well. While Choice cooperates with these partners, it also competes with them, notes Fletcher.

“The ecosystem or the set of applications



that manage our existing suite of distribution channels is quite complex," says Fletcher. "It has grown over the course of 15 years. When the first GDS, OTA, and online travel agencies started popping up, we needed to allow people to book against our inventory systematically. The technological landscape was very different back then, and there's always been

a lot of pressure to deliver on those systems, so they haven't necessarily gotten the care and feeding that they should have."

When the Google opportunity came along, however,

PEAK PERFORMANCE

Choice's Google Interface service handles up to 750,000 transactions a day and is capable of a peak rate of 540,000 transactions per hour.

More Than an App: AN ARCHITECTURE

given the lodging company a new standard for its IT systems architecture: Java.

Before the Google interface project, Choice's Phoenix, Arizona-based IT operations didn't have an overall systems architecture or even a systems architecture department, says Rain Fletcher, the company's vice president, application development and architecture. "There was never anybody who said, 'This is how we're going to develop software,' so developers were left to their own devices," Fletcher explains.

That approach may have worked in the past, but today Choice faces a complex, fast-changing business environment, fueled in large part by the Web and mobile devices. To stay on top, the company needs core IT systems that can adapt just as quickly.

To bring its systems up to date, Choice is migrating from its current legacy environment to a new service-oriented architecture (SOA), in which business processes are packaged as discrete reusable software components or enterprise services.

Because of the Google project, Choice's new SOA is based on Java, including the language and its related frameworks and APIs. In addition to the benefits of portability, scalability, and high performance, Java is especially suited to deliver on the promise of SOA environments. "Java has great support for SOA-based languages such as XPath, XML, and XSLT," notes Fletcher.

The proof is in the payoff. Choice is gaining four reusable enterprise services from the Google project: hotel distribution information, room availability, rate changes, and inventory. Two of these services—hotel distribution and room availability—are already being reused to power the company's push into mobile devices with its Choice Mobile application for Apple iPhone and Android-based devices.

The Google Interface project has given Choice Hotels International more than a new and innovative way to market its rooms to potential guests. It has also

Choice's IT staff quickly realized it had to think outside the box. "We didn't want to deliver the Google Interface on the existing platform," explains Fletcher. "The existing tools that we had used over the last 15 years to build those interfaces would have supported this also, although it would have been quite complex because of the unique requirements that Google presented. But we

really didn't want to do that; we wanted to embrace our new architecture and start to abstract those other systems so that, hopefully, we can retire them one day soon."

Indeed, the set of Java tools, standards, and frameworks the Choice IT team put together to develop the Google Interface has become the company's IT architectural standard.

THE STANDARD OF CHOICE

The Google team had three unusual technical requirements for Choice's new hotel information service. First, they wanted Choice to maintain the cache of data on Google's computers and systems, rather than feed it out to Google to cache. Second, Google wanted only a rolling 90-day supply of room and rate information—a mere subset of the data in Choice's CRS. And third, the information had to be updated every seven days or less, whether it had changed or not. "They assumed that the cache was stale after seven days, and they would stop exposing that rate and availability information," explains Fletcher.

Executives on the business side at Choice saw the project as an intriguing extension of Choice's distribution platform. "All of a sudden this was the most important project in the building," says Fletcher, adding that existing staff was requisitioned to work on the project in addition to their current workloads.

With all hands on deck and the support of Java and the Java platform, the Google Interface went live in mid-October last year. Today it is available to any Google Maps or Google Places user and can handle up to 750,000 transactions per day.

The Google Interface consists of three components, all written in Java. One component is a custom Java EE Connector Architecture (JCA) adapter that interfaces with Choice's CRS. The second component decides which information needs to get pushed out to Google, and the third, a scheduler, guarantees that the published data is never more than seven days old.

Scalability and performance were important considerations for choosing Java, but Choice had other reasons as well, says Fletcher. With Java, the company would achieve both hardware and software vendor independence. Choice already had Java expertise in-house—indeed, 80 percent of its IT staff already had some Java knowledge. Java talent is also

widely available in the marketplace. Most important of all is Java's status as a premier language and development platform for enterprise applications.

“As a language, it is mature, well understood, and has a robust and expressive suite of powerful APIs,” says Fletcher. The Google Interface project, in fact, uses a number of those APIs, including JCA and the Java Message Service (JMS).

“We made the decision to standardize on Java as our standard development platform specifically because of projects such as our Google Interface,” Fletcher concludes, “which required complex logic, very high performance, and the ability to get to market quickly.” [</article>](#)

Philip J. Gill is a San Diego, California–based freelance writer and editor.

SPEEDY DELIVERY
Choice Hotels
had less than
three months
to develop its
Google Interface
service.

Data Quality Tools for Java



BEFORE

john smith iii phd
melissa data corp.
22382 Empresa 92688
7145895200
john@800miAL.con

AFTER

Melissa Data Corp.
John Smith III PhD
22382 Avenida Empresa Ste 100
Rancho Santa Margarita, CA 92688-2112
949-589-5200
John@melissadata.com
Delivery Indicator: Business

*Highlights indicate added and/or corrected data.

Realtime NCOA^{Link}
Change-of-Address
Web Service
available

Now, finding the right data verification tools doesn't have to be so puzzling. Melissa Data offers customizable APIs, Web services and enterprise applications to match your budget and business needs. For solutions to cleanse, validate and standardize your contact data, we're ready to help you find the perfect fit.



Request free trials at
MelissaData.com/myjava or call 1-800-MELISSA

- Global address verification for 240 countries
- Clean and validate data at point-of-entry or in batch
- Correct misspellings, missing directionals, and confirm deliverability
- Enhance addresses with County, Census, FIPS, etc.
- Append rooftop lat/long coordinates to street addresses
- Update records with USPS and Canadian change of address info

MELISSA DATA[®]

Your Partner in Data Quality



Improving and extending the code leads to the first playable version of our turtle game.

The turtle's `act` method, as we left it last time, is shown in **Listing 1**. Before we go on to add more functionality to our program (and, in the process, make our code more complex),

Each of the method definitions has a section

Note that this version of our code does exactly the same thing

Compile your scenario and place a few snakes into your world (see **Figure 2**). You might want to save the world again (right-click the world background and choose **Save the World**) to add your snakes to the saved world.

PHOTOGRAPH BY JOHN BLYTHE

We can start by copying all three methods from the `Turtle` class (`act`,



At this point, you might like to compile and run your scenario to test what it does. It's not exactly what we want yet, but it does something. (You will see that the snakes follow your key presses, just as the turtle does.)

There are two things we'd like to change in the class `Snake` from the copied turtle behavior. First, snakes should eat turtles, not pizza. And second, snakes should move around on their own (randomly), rather than be controlled by the keyboard.

We can make the first change by going to the snake's `eat` method and replacing all mentions of "pizza" with "turtle," as shown in **Listing 3**.

Pay attention to capitalization: Uppercase and lowercase letters must be used exactly as shown. When writing *Turtle* (with a capital *T*), we are referring to the **Turtle** class. When writing in lowercase (*turtle*), we are referring to a *variable*.

LISTING 3

```
public void act()
{
    move(4);

    if(Greenfoot.isKeyDown("left")) {
        turn(-3);
    }

    if(Greenfoot.isKeyDown("right")) {
        turn(3);
    }

    Actor pizza = getOneIntersectingObject(Pizza.class);
    if(pizza != null) {
        getWorld().removeObject(pizza);
    }
}
```

 [See all listings as text](#)



Real-Life Application for JAX-WS Web Services with NetBeans 7.0.1

We will use the NetBeans IDE for developing a typical enterprise application that will show you how to access and consume a JAX-WS Web service (an external service that resides in the application tier) from JSF client applications.

- Add a new JAX-WS Web service that extrapolates the amount of the bid.
- Generate the WSDL file.
- Create a Web service client to consume the

Unlike RESTful Web services, **JAX-WS Web services** are generally used for larger applications that demand several channels of communication among platforms, and they provide tools for integration and interoperability.

Let's Add and Test the JAX-WS Web Service

1. Add the new Web service in the AuctionApp application:

- Open the AuctionApp in NetBeans IDE version 7 or later.
- Right-click the **AuctionApp** project note and choose **New**. Then select **Web Service**.

- c. Type **AuctionAppSOAPws** in the **Web Service Name** field and type **com.bonhbel.oracle.auctionApp.ws** in the **Package** field. Then click **Next**.

- d. Select the **Implement Web Service as a Stateless Session Bean** box in the upper part of the New Web Service dialog box.

- e. Click Finish.**

NetBeans generates the structure for the new Web service with a sample Web service and all the resources we need to operate our Web service.

At this point, the source code for the Web service can be edited in the editor. Your AuctionAppSOAPws.java file should look like the code shown in **Listing 1**.

LISTING 2

```
package com.bonbhel.oracle.auctionApp.ws;

import javax.ws.WebService;
import javax.ws.WebMethod;
import javax.ws.WebParam;
import javax.ejb.Stateless;

/**
 * @author Max Bonbhel
 */
@WebService(serviceName = "AuctionAppSOAPws")
@Stateless()
public class AuctionAppSOAPws {
    /** This is a sample web service operation */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```



[See all listings as text](#)

2. Now add a specific operation to the Web service that will extrapolate the amount of the bid by item:
 - a. Open the **Web Services** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node. Then select **Add Operation**.
 - b. Type **extrapolateAmountBid** in the **Name** field and type **double** in the **Return Type** field.
 - c. Click **Add** in the lower part of the Add Operation dialog box.
 - d. Type **amountBid** in the **Name** field and type **double** in the **Type** drop-down list.
 - e. Click **Add** again.
 - f. Type **factor** in the **Name** field and type **int** in the **Type** drop-down list.

- g. Click OK.
3. Edit and modify the generated operation, as shown in **Listing 2**.
4. Test the JAX-WS Web services:
 - a. Right-click the **AuctionApp** project and choose **Deploy**.
 - b. Expand the **Web Service** node of the AuctionApp project. Then right-click the **AuctionAppSOAPws** node and choose **Test Web Service**.

A tester page is displayed in your browser. If you see the page shown in **Figure 1**, it means your JAX-WS Web services are working correctly.
 - c. Type two numbers in the tester page: **80** for **amountBid** and **100** for **factor**.
 - d. Click **extrapolateAmountBid**.

JAX-WS allows us to develop and deploy Web services easily **using the Java platform.**

It's time to create a Web client to consume the JAX-WS Web services we created. So we are going

Let's code this application in five minutes.

- g.** Click **Finish**.
- 2.** Create the entities based on the datasource we created in Part 1 of this series:
 - a.** Right-click the **AuctionAppWebServiceClient** project and select **New**. Then select **Entity class from Database**.
 - b.** In the Databases Tables dialog box, select your datasource in the **Data Source** drop-down list.
 - c.** In the upper part of the Databases Tables dialog box, select the **BID**, **SELLER**, and **ITEM** tables and click **Next**.
 - d.** In the Entity Classes dialog box, type a **Package name**, such as **com.bonbhel.oracle.auctionAppWebServiceClient**.
 - e.** Accept all other default settings, and then click **Finish**.NetBeans generates the Seller.java, Item.java, and Bid.java files.
- 3.** The client implementation we are going to build consists of JSF pages based on the entities just created, so create the JSF pages:
 - a.** Right-click the **AuctionAppWebServiceClient** project and select **New**. Then select **JSF Pages from Entity Classes**, click **Add all**, and click **Next**.
 - b.** Type a **Session Bean Package name**, such as **com.bonbhel.oracle.auctionAppWebServiceClient.facade**, and type a **JSF Classes Package name**, such as **com.bonbhel.oracle.auctionAppWebServiceClient.controller**.
 - c.** Type a **Folder Name**, such as **jsfClient**, and click **Finish**.

- f.** Accept all other default settings.
The package name will be taken from the WSDL file.
 - g.** Click **Finish**.
NetBeans generates the structure of the new Web service client, as shown in **Figure 3**.
- Note:** If the WSDL file cannot be downloaded, click **Set Proxy** and then specify the proxy server.



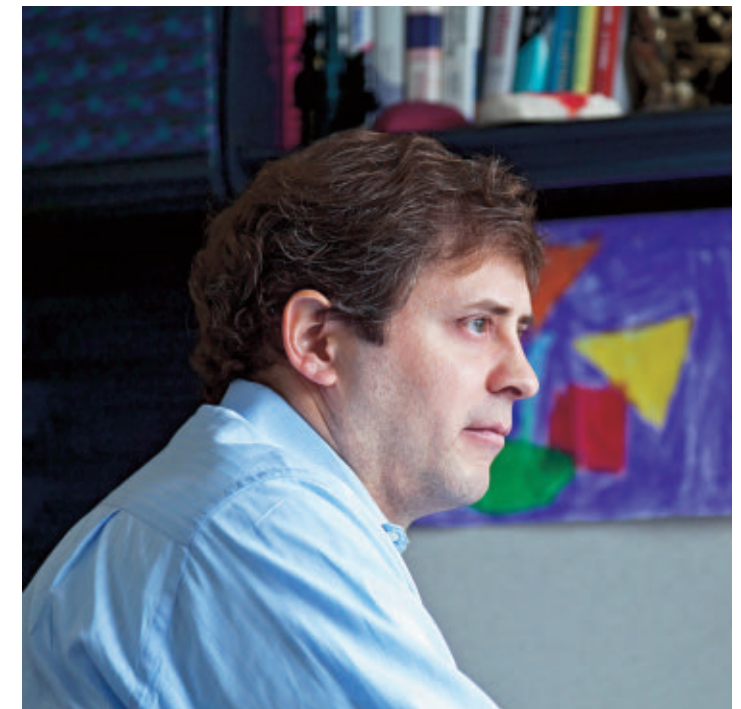


Cloud/Java EE

Tame the Cloud

Oracle's **Cameron Purdy** on how Java EE is evolving to become the cloud computing platform standard. **BY MICHAEL MELOAN**

The enterprise landscape is increasingly complex, with information flowing from more sources than ever before. The emphasis is shifting from standalone applications to dynamic shared environments that will enhance scalability, automation, and efficiency. Cloud computing promises to be a transformational technology for Java EE. Cameron Purdy, vice president of Java EE development at Oracle, provides an overview of the key issues in this arena and how they will affect enterprise developers and customers.



Java Magazine: How will cloud computing deliver advantages in the enterprise space?

Purdy: At this fairly early stage in the development of cloud technology, the major focus is on PaaS [platform-as-a-service] capability. A number of infrastructure cloud vendors offer dozens or even hundreds of servers as easily as buying something on an e-commerce site. This is a move toward decentralization and democratization of IT infrastructure. It offers abstraction of hardware and software elements all the way up to development environments and middleware components, which

PHOTOGRAPHY BY DAVE BRADLEY
ART BY I-HUA CHEN

PRIMED FOR CLOUD
The container-based architecture and its inherent abstraction of resource access make the Java EE platform **well suited for cloud computing.**

freedom allows developers to focus on enterprise solutions. This approach offers real benefits in terms of operational simplicity, time to market, and cost savings.

At Oracle, we'll be looking at how PaaS will affect different roles within an organization, such as development, QA, operational staging, and actual production operations. Then we need to integrate these considerations into our own suite of products, as well as work with our industry partners to standardize the model, the APIs, and the overall Java EE platform specifications.

Java Magazine: Can you provide an example of how Oracle is reflecting these consider-

ations in its products?

Purdy: Sure. For example, within Oracle WebLogic Server 12c, we provide support for dependency injection, the Java Persistence API [JPA], and a uniform build process via the [Maven WebLogic Plug-in](#), as well as for Agile development methodology in concert with [Hudson Continuous Integration](#). These standards make enterprise applications easier to build as well as operate. Significant improvements have also been realized in managing and achieving high performance in virtualized environments, which is a prerequisite in PaaS environments. At Oracle, we deliver these optimizations with Oracle VM.

Java Magazine: What are the most important aspects to consider when implementing a cloud-oriented enterprise architecture?

Purdy: There are three key concepts related to deploying enterprise PaaS solutions: availability, elasticity, and multitenancy. In general, applications will be running on virtual machines in a remote environment controlled through a Web-based console. Because there is no physical access to a local system, the ability to mitigate problems is dramatically reduced. To ensure availability, systems need to be more autonomic. They need to be self-healing. One of the ways to achieve this resiliency is through redundancy.



(From left to right) Oracle's Mac Hale, director of software development; Cameron Purdy, vice president of Java EE development; and Jim Gish, consulting member of technical staff, talk shop.



Oracle's Mike
Lehmann and
Arun Gupta
discuss Oracle
WebLogic Server
12c at Oracle
OpenWorld Latin
America 2011 in
São Paulo, Brazil.

Maintaining availability, even in the event of hardware failure or electrical failure, requires that the application be running in multiple places, and possibly in multiple data centers, to prevent a single point of failure.

Availability is also facilitated through the concept of elasticity. Applications need to be robust enough to seamlessly operate while a given server is down for maintenance or due to failure. They must also be able to respond to additional capacity demands from increases in concurrent users or fluctuations in processing load. Elastic applications must be able to add resources dynamically, which fits very well with the cloud computing metaphor.

The concepts of availability and elasticity are both in concert with enterprise architectures built around PaaS and IaaS [infrastructure as a service].

Multitenancy is also an important principle in cloud computing. It refers to the ability to collocate multiple users within a single environment. A CRM [customer relationship management] system, for example, would need to provide service to a number of different companies. Each company represents a different tenant. And those tenants could be supported by dedicated hardware, or virtualized, providing each tenant with a virtual machine or a set of virtual machines within a given hardware infrastructure. There are many levels of multitenancy, even down to multiple tenants inside a single JVM [Java Virtual Machine].

There are trade-offs across the board in multitenant architectures. Security, resource utilization, and cost issues are at the forefront. Creating isolation and protection within a JVM is an important area of investment right now. We are also looking at how to meter CPU

and memory demands by a particular tenant. Managing and restricting how many resources a tenant can consume is one of the fundamental challenges of multitenant systems. We need to make sure that a tenant can't chew up an entire CPU or use so much memory that a system is brought to its knees. We want to provide building block technologies to manage these issues for our customers.

As we revisit each aspect of Java EE, multi-tenancy, availability, and elasticity will all be important considerations in planning with our partners for the future of the platform.

Java Magazine: What new capabilities will the Java EE 7 release offer?

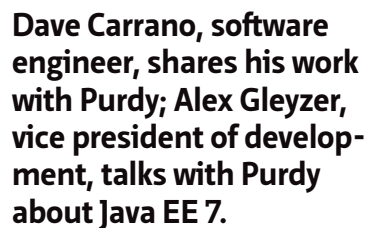
Purdy: We are working very closely with our industry partners to define and deliver Java

EE 7. The major theme in Java EE 6 was ease of use, and that continues in Java EE 7. But the prime mover for Java EE 7 will be the cloud. The container-based architecture and its inherent abstraction of resource access make the Java EE platform well suited for cloud computing. This approach ensures that portable applications can target single-machine deployments as well as large cluster installations without fundamental changes to the programming model. The move to cloud computing is a further evolution of this powerful paradigm.

Multitenancy will be supported via virtual servers mapping requests to a tenant. And there will be more emphasis on CDI [Contexts and Dependency Injection], the JPA,



In Oracle's Burlington, Massachusetts, office, Oracle Architect Gene Gleyzer updates Purdy on his current projects.



Modularity and versioning capabilities are also being investigated. This work will be coordinated with the upcoming modularity

Regarding lambda expressions, they're typically useful in implementing aggregate operations that can frequently be parallelized using multicore processors. We've seen opportunities for leveraging this in technologies like JDBC, JPA, EJBs, JMS filtering, and many other areas. Lambda expressions have been powerful in other languages and envi-

I expect to see wonderful levels of innovation as we begin to explore the possibilities of HTML5 and cloud technology. It's incredibly exciting to be a part of it. </article>

Michael Meloan began his professional career writing IBM mainframe and DEC PDP-11 assembly languages. He went on to code in PL/I, APL, C, and Java. In addition, his fiction has appeared in *WIRED*, *BUZZ*, *Chic*, *LA Weekly*, and on National Public Radio.

- [Java EE at a Glance](#)
- [Oracle WebLogic Server](#)

- Status:
 - Approved by the JCP
 - Spec leads: Yannis Cosmadopoulos (Oracle), Cameron Purdy (Oracle), and Gregory Luck (Software AG)
 - [Project page](#)
 - [Mailing list archive](#)

- API that can be used by applications and Java EE containers to offload the responsibility of statement management to third-party providers with different QoS characteristics
- Java SE-based callers can access the state data by querying the state providers
- Providers with different QoS can be added, and API callers can query to meet their criteria
- Package: javax.state and javax.state.provider
- Status:
 - Approved by the JCP
 - Spec lead: Mitch Upton (Oracle)
 - [Project page](#)
 - [Mailing list archive, jsr-350-experts@java-state-management.java.net](#)

- Programming model for batch applications and a runtime for scheduling and executing jobs
- Defines Batch Job, Batch Job Step, Batch Application, Batch Executor, and Batch Job Manager for the standard programming model
- Package: `javax.batch`



Project Coin: The Java Language Has Evolved!

Project Coin is an OpenJDK project that initiated the Java language changes that were standardized as part of Java SE 7 under [JSR-334](#) in the Java Community Process (JCP). The project is self-described as follows:

- *Strings in switch*
- *Binary integral literals and underscores in numeric literals*
- *Multi-catch and more precise rethrow*
- *Improved type inference for generic instance creation (diamond)*
- *try-with-resources statement*
- *Simplified varargs method invocation*

each feature, and also to provide some background on the implications for implementing each of them, especially since they all maintain backward compatibility with prior versions of Java SE. The impact of each change was

remarkably balanced with a scientifically sound analysis of millions of lines of existing Java code.

Part 1 focuses on the addition of binary integral literals, underscores in numeric literals, strings in `switch` statements, and type inference for generic instance creation.

Note: The source code for the examples described in this article can be downloaded [here](#).

Prior to Java SE 7, numeric literals could be specified either in *decimal*, *octal*, or *hexadecimal* forms, such as in the following:

The rules are simple. A decimal number is a literal comprising a combination of digits between 0 and 9, an octal number starts with a zero, and a hexadecimal number starts with **0x**. With the advent of Java SE 7, you can now specify a number as a binary literal by starting it with **0b** followed

by a combination of zeros and ones up to the size of the underlying primitive type. The decimal number 123 can be written as the following binary literal:

```
// 123 in binary
byte binary = 0b01111011;
```

If fewer digits are provided than expected for the underlying primitive type, the missing digits are considered to be at the beginning of the representation, and they are given a value of zero. Thus, the



Julien Ponge provides a brief introduction to his article about Project Coin.

previous example could be written with one less digit:

```
// 123 in binary
byte binary = 0b1111011;
```

There are arguably a few readability problems in literals as the number of digits increases. While hexadecimal and binary literals start with a clear prefix (**0x** and **0b**), octal literals start with a zero, which can be confusing. Indeed, it is naturally tempting to interpret 0173 as 173 instead of 123. In a bid to increase readability, Java SE 7 adds the ability to introduce underscores as part of numeric literals. As an example, 1200645790 can be written as 1_200_645_790. Back to the examples based on the number 123, we can rewrite them as follows:

```
byte decimal = 123;
byte octal = 0_173;
byte hexadecimal = 0x7B;
byte binary = 0b0111_1011;
```

Of course, underscores can also be used in **float**, **double**, and **long** numeric literals. A double number ends with **D**, while a long integer ends with **L**. Thus, the following literals can benefit from underscores:

```
double doubleValue = 1.111_222_444D;
long longValue = 1_234_567_890L;
long longHexa = 0x1234_3b3b_0123_cdefL;
```

There are, however, certain rules to respect in the usage of underscores. While they can be repeated at will

(1____2__3 is the same as 123 or 1_2_3), they can appear only between digits. They cannot be put at the beginning or at the end of a numeric literal, before or after the point in a floating-point number, in or after a binary or hexadecimal literal prefix, and before or after a **long** (**L**) or **double** (**D**) suffix. Hence, the following literals raise a compiler error:

```
int a = _123_;
long b = 123_L;
int c = 0x_abc;
int d = 0_x_abc;
double e = 1000_._666_666_D;
```

Possible correct literals would be

```
int a = 123;
long b = 1_2_3L;
int c = 0xa_bc;
int d = 0xa_bc;
double e = 1000.666_666D;
```

This change in the specifications of the Java language is transparent to the Java Virtual Machine and existing bytecode and libraries, because the Java compiler simply ignores underscores. Similarly, the new binary literals are treated as a new case beside octal and hexadecimal literals.

In summary, these changes are lightweight yet beneficial for improving the readability of code manipulating numeric literals.

Strings in Switch Statements

In the Java programming language, the **switch** control-flow statement works

LISTING 1 LISTING 2 LISTING 3

```
private static void basicSwitch(int value) {
    switch (value) {
        case 2:
            System.out.println("Number 2!");
        case 1:
        case 3:
            System.out.println("A number in the [1, 3] range: " + value);
            break;
        default:
            System.out.println("A number outside the [1, 3] range: " + value);
    }
}
```



[See all listings as text](#)

with integer values, such as in **Listing 1**.

In **Listing 1**, a message is printed for cases where **value** is equal to 1, 2, or 3. A special case exists when **value** is 2, because a message is printed before passing over the execution to the handling cases of values 1 and 3. Other cases are collected as part of the **default** case. The **switch** statement is not limited to **int**. It also works for **char**, **byte**, and **short**, as well as the object wrapper types **Character**, **Byte**, **Short**, and **Integer**. Java SE 5 introduced the support of enumerated types, and they can also be used in **switch** statements. **Listing 2** illustrates switches on enumerated types.

Switching on enumerated types is based on the value returned by the **ordinal()** method of **java.lang.Enum**, the base class of all enumerated types, and that returns an integer. Hence, such a **switch**

statement is still based on an integer value that is arbitrarily assigned by the Java compiler to each enumerated value.

Java SE 7 allows strings to be used in **switch** statements, too. This is interesting, because strings are often used in conditions for control-flow branching.

Listing 3 is an example reflecting the use of strings in **switch** statements.

A **NullPointerException** is thrown if the value to be switched on is the **null** value. Also, the **case** statements need to be on string literals or any constant expression of **java.lang.String** type.

The addition of strings in **switch** statements did not require any modification to the Java Virtual Machine bytecode. Rather, this functionality was implemented as syntactic sugar, where the compiler infers semantically equivalent code, but it is based on integer switches.

In order to understand how this works, we can use the `javap` decompiler tool, which ships as part of the Java SE Development Kit, and analyze the generated bytecode for the `isTrue()` method above. The equivalent Java code is shown in **Listing 4**.

Java compilers are free to implement `switch` statements with strings differently as long as the semantics are preserved. The string switched on is compared for equality against the `case` label. The generated code starts by a first `switch` statement based on the `hashCode()` value of the string constants being used. This acts as a way to directly jump to the comparison between the switched string and the possible `case` constants.

Note that the `hashCode()` value alone is not enough to compare strings, because the risk of collisions between different strings having the same `hashCode()` value is too high. It is even relatively easy to construct a string with a given `hashCode()` value (see Joe Darcy's post, [Unhashing a String](#)). When a string is matched, an integer is set based on the index of the original `case` statement, and it is used in the subsequent `switch` statement that contains the original instructions intended for the string `case` statements.

Improved Type Inference for Generics

Generics were added in Java SE 5 to increase the type safety in programs. They are especially useful when dealing with Java collections. **Listing 5** shows a typical usage of generics on collections.

There is, however, a strong sense of ceremony in this code. Let us just

consider the declaration of `strings` and replace the code with the following:

```
List<String> strings =  
new LinkedList<Integer>();
```

Of course, this code does not compile. There is a necessary concordance between the parametric types in a variable declaration and instance creation. Having to repeat explicit types on each side of such assignments is considered to be redundant by most developers.

Java SE 7 introduces a new syntax for calling the [new](#) operator, which is called *diamond* or simply "<>." It is used in place of a parametric type argument list such as `<String, List<String>>` when we instantiated `contacts` in the example above. The compiler interprets it as a point where the types shall be inferred from the context and usage.

Back to the declaration of `strings` of type `List<String>`, it is easy to infer that the type for a `LinkedList` to be assigned must resolve as `LinkedList<String>`. The code above can thus be rewritten as shown in **Listing 6**.

The change is marginal when a single parametric type is used, such as for strings, but it becomes much more interesting when more of them are involved, especially when they also require nested parametric types, such as for the declaration of `contacts`.

Diamond may be used with spaces between `<` and `>`, but this is discouraged:

```
List<String> strings =  
new LinkedList<>();
```

LISTING 4 LISTING 5 LISTING 6

```
public static boolean isTrue(String s) {
    String str = s.trim().toUpperCase();
    int jump = -1;
    switch(str.hashCode()) {
        case 2404:
            if (str.equals("KO")) { jump = 3; }
            break;
        case 2497:
            if (str.equals("NO")) { jump = 4; }
            break;
        case 2524:
            if (str.equals("OK")) { jump = 0; }
            break;
        case 87751:
            if (str.equals("YES")) { jump = 1; }
            break;
        case 2583950:
            if (str.equals("TRUE")) { jump = 2; }
            break;
        case 66658563:
            if (str.equals("FALSE")) { jump = 5; }
        }
    switch(jump) {
        case 0:
        case 1:
        case 2:
            return true;
        case 3:
        case 4:
        case 5:
            return false;
        default:
            throw new IllegalArgumentException("Not a valid true/false string.");
    }
}
```

 [See all listings as text](#)

It must be noted that a type declaration involving diamond should not be confused with a raw type. Hence, an instance created with `new LinkedList<>()`

does not have the same type as one created with new `LinkedList()`.

As with all good things, certain restrictions limit the applicability of generic

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2012



Creating lots of interesting effects is effortless in Java SE 7 thanks to the new JLayer component in Swing.

At the end of the day, we had a book full of hacks—fun code snippets that did interesting things but could break at any moment. They were great for playing around but not good enough for heavily used production code.

At its most basic, **JLayer** is a Swing container. It holds regular Swing com-

- Creates a button



- Figure 1** shows transparent red drawn on top of a control.

Notice that the component to be drawn on is included as an argument to the `paint` method. This is very important. The `LayerUI` is typed to the component it will



So far we've used **JLayer** for some trivial overdriving effects, but there are many

more-useful things you could do with it. Let's look at a few.

When trying to debug applications it's common to want to know the bounds of every component in the frame, for example, when one of your components seems to be invisible. While you could print this information to the console, with `JLayer` you can simply draw debugging information directly on top of the components.

The example shown in **Listing 5** and **Figure 3** draws the borders of every component in orange. It also writes the name of the component's class. This is handy if you want to tell the difference between many button subclasses.

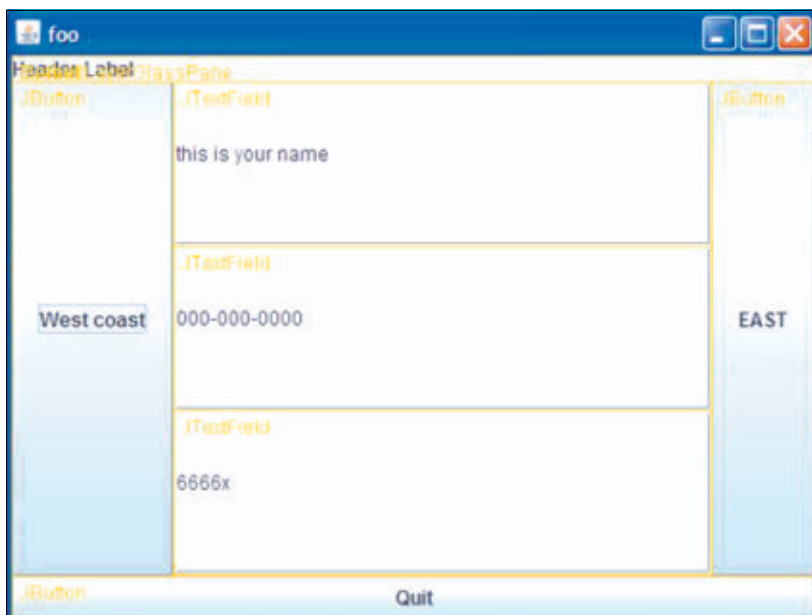


Figure 3

GOING DEEP

The `JLayer` is created only once and set on the topmost panel. Because the `LayerUI` is recursive, it can draw on top of child components as deep in the hierarchy as necessary.

The code in **Listing 5** again creates a `LayerUI` instance with a `paint` method. This time it recursively goes through the component and its children to draw borders. It also writes out the simple name of the component's class if the component doesn't have any children. (It skips components with children to avoid overdraw on nested panels.)

Note that the `JLayer` is created only once and set on the topmost panel. Because

the `LayerUI` is recursive, it can draw on top of child components as deep in the hierarchy as necessary.

The code in **Listing 6** is similar to what we've done before, but it draws any disabled component using a blur. That way, the user knows it is really disabled.

In this case, if the component is enabled, `super()` is called to draw normally. If the component is *not* enabled, the code creates an image buffer and calls `super()` on the graphics context from the image. This draws the component into the image instead of onto the screen. Then the code applies a simple blur operation to the image and draws

LISTING 5

LISTING 6

```
public static void main(String ... args) {
    LayerUI<JPanel> overlay = new LayerUI<JPanel>() {
        @Override
        public void paint(Graphics g, JComponent clayer) {
            super.paint(g, clayer);
            drawBorders(g, clayer);
        }
        private void drawBorders(Graphics g, JComponent clayer) {
            g.setColor(Color.ORANGE);
            g.drawRect(0, 0, clayer.getWidth(), clayer.getHeight());
            if (clayer.getComponentCount() <= 0) {
                g.drawString(clayer.getClass().getSimpleName(), 5, 15);
            }
            for (Component comp : clayer.getComponents()) {
                if (comp instanceof JComponent) {
                    g.translate(comp.getX(), comp.getY());
                    drawBorders(g, (JComponent) comp);
                    g.translate(-comp.getX(), -comp.getY());
                }
            }
        }
    };
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.add(new JLabel("Header Label"), BorderLayout.NORTH);
    panel.add(new JButton("Quit"), BorderLayout.SOUTH);
    panel.add(new JButton("EAST"), BorderLayout.EAST);
    panel.add(new JButton("West coast"), BorderLayout.WEST);
    JPanel subpanel = new JPanel();
    subpanel.setLayout(new BoxLayout(subpanel, BoxLayout.PAGE_AXIS));
    subpanel.add(new JTextField("this is your name"));
    subpanel.add(new JTextField("000-000-0000"));
    subpanel.add(new JTextField("6666x"));
    panel.add(subpanel, BorderLayout.CENTER);
    JFrame frame = new JFrame("foo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(new JLayer(panel, overlay));
    frame.pack();
    frame.setVisible(true);
}
```



[See all listings as text](#)

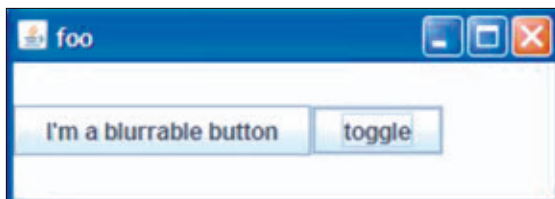


Figure 4

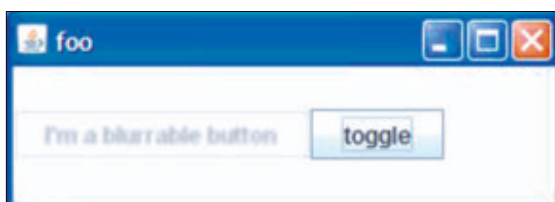


Figure 5

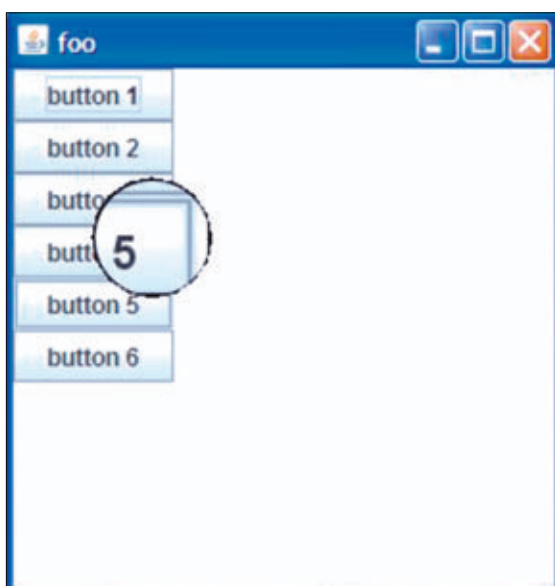


Figure 6

that final image to the screen. **Figure 4** shows an example of a button that can be blurred, and **Figure 5** shows a blurred button.

Because the `JLayer` repaints any time its view control changes, we don't need to track any state. A toggle button can simply toggle the `enabled` property of the button, and Swing takes care of the rest.

As a final example, **Listing 7** shows how you can do something completely

crazy with **JLayer** by building a magnifying glass that will magnify whatever is underneath the mouse as the mouse moves, as shown in **Figure 6**.

The code shown in **Listing 7** is roughly the same as before. The difference is that this time it scales the graphics *before* drawing into an image. Then it draws the image back to the screen with a nice round clip.

Drawing components into a temporary image is a really useful technique because there are so many ways to post-process an image. To further extend the magnifying glass, you could add a distortion filter to give the scaled image a rounded effect.

Conclusion

JLayer is a very powerful addition to Swing that makes many previously hard or impossible effects easy to do in a clean and supported way. **JLayer** proves that Swing still has a lot of life left in it, and Java SE 7 provides new ways of extending Swing to do interesting things. (And being more generified is always nice, too.)

The examples shown here are just the tip of the iceberg. You could use `JLayer` to implement status overlays, busy indicators, global right-click menus, and much more. `</article>`

LEARN MORE

To learn more about **JLayer**, read the Java docs for **JLayer** and **LayerUI**:

- Layer
- LayerUI

LISTING 7

```
@Override
public void paint(Graphics g, JComponent c) {
    //draw regular
    super.paint(g,c);
    //draw into buffer
    Graphics2D g2 = img.createGraphics();
    int ih = img.getHeight();
    int iw = img.getWidth();
    //fill with white
    g2.setPaint(Color.WHITE);
    g2.fillRect(0, 0, img.getWidth(), img.getHeight());
    //render view component scaled and translated
    Graphics2D g3 = (Graphics2D) g2.create();
    g3.translate(-pt.x + iw, -pt.y + ih*2);
    g3.scale(2, 2);
    g3.translate(-pt.x/2 - iw/4, -pt.y/2 - ih/4);
    super.paint(g3,c);
    g3.dispose();
    //draw black border
    g2.setPaint(Color.BLACK);
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);
    g2.setStroke(new BasicStroke(3));
    g2.draw(new Ellipse2D.Double(0,0,iw,ih));
    g2.drawRect(0,0, iw-1, ih-1);
    g2.dispose();
    //draw image to screen
    Shape oldClip = g.getClip();
    int mx = pt.x-img.getWidth()/2;
    int my = pt.y-img.getHeight();
    g.setClip(new Ellipse2D.Double(mx,my,iw,ih));
    g.drawImage(img, mx, my, null);
    g.setClip(oldClip);
}
```

 [See all listings as text](#)



Understanding the Hudson Plug-in Development Framework

- **entry** tells Hudson that the enclosed tags are considered to be user interaction elements and they are submitted via an HTML form.

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2012

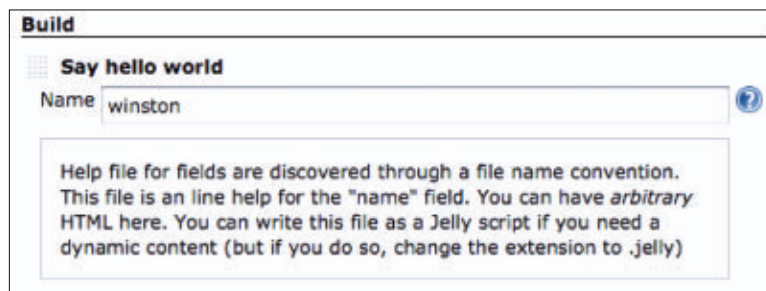


Figure 1

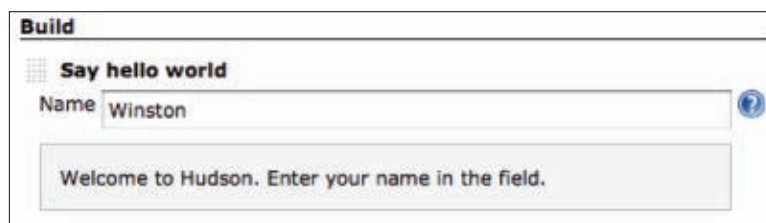


Figure 2

- `textbox` renders a simple HTML text field whose value will be sent back to the server.

Understanding the UI Rendering

Let's take a closer look at how the UI is rendered from the Jelly file. In Part 1 of this series, we created a Hudson project called [TestProject](#). Open the [TestProject](#) job configuration page by clicking the [Configure](#) link. Scroll down to the [build](#) section and view the [Say Hello World](#) builder and its configuration. Click the [Help](#) button ("??") on the right side of the text box. It displays online help text, as shown in **Figure 1**.

Let's find out where this help text comes from. The content of `config.jelly` has no such help text. Once again, convention comes into play. The folder containing the config file also has a file named `help-name.html`. Open the file using an editor and notice that it contains the exact help text shown in **Figure 1**.

How does Hudson know to get the content from this file and display it as help content for the field? The trick is in the name of the file. By convention, Hudson looks for a specific filename in the same folder that contains the config file. The name of the file should be `help-fieldName.html`.

In the config.xml file we have the following:

```
<f:entry title="Name"
field="name">
<f:textbox />
</f:entry>
```

`field="name"` indicates that the text box should be used as an entry field with the name `Name`. So, based on convention, the help text for that field should exist in a file named `help-name.html`.

The content of the `help-name.html` file is pure HTML. You can include images, text, and hyperlinks in the content to emphasize and enhance your help text. As mentioned in the example help text in **Figure 1**, if you want to use information from Hudson model objects, you should have Jelly content in the field help file and the file extension should be `.jelly` instead of `.html`. **Listing 2** shows an example file called `help-name.jelly`.

WHAT'S INSIDE?

The **content** of the help-name.html file is **pure HTML**; you can include images, text, and hyperlinks.

LISTING 1

LISTING 2

```
<j:jelly xmlns:j="jelly:core" xmlns:st="jelly:stapler" xmlns:d="jelly:define"
xmlns:l="/lib/layout" xmlns:t="/lib/hudson" xmlns:f="/lib/form">
<!--
Creates a text field that shows the value of the "name" property.
When submitted, it will be passed to the corresponding constructor parameter.
-->
<f:entry title="Name" field="name">
<f:textbox />
</f:entry>
</j:jelly>
```



[See all listings as text](#)

When Hudson is stopped and restarted with the above Jelly help file, clicking the Help button for the same text field displays the text shown in **Figure 2**.

Note that `_${app.displayName}` is replaced with `Hudson`, which is the name of the application.

UI and Model Object Interaction

Now let's see how the UI interacts with Hudson model objects. **HelloBuilder** is a Hudson model object. It encapsulates data. The UI can interact with this model to get and display its data, get information from the user via fields in the UI, and update the model data.

The help file, `help-name.jelly`, contains a JEXL expression ``${app.displayName}`. When the server side of the Hudson application receives the request to render the job configuration page, it includes both the snippets `config.jelly` and `help-`

name.jelly in the job configuration page, which itself is another Jelly file.

The entire Jelly file is given to the Jelly renderer to render into the client-side code. The Jelly renderer is responsible for substituting the corresponding value for the JEXL expression after evaluating the expression. The first part of the expression evaluates to the model object and then to the method name of the model object. By default, Hudson registers three identifiers for the model objects to the JEXL expression evaluator:

- **app**, which is the Hudson application itself. For example, ``${app}.displayName`` evaluates to `Hudson.getDisplayName()`.
- **it**, which is the model object to which the Jelly UI belongs. For example, ``${it.name}`` evaluates to `HelloWorldBuilder.getName()`.
- **h**, which is a global utility function called [Functions](#) that provides static utility methods. For example, ``${h.clientLocale}`` evaluates to `Functions.getClientLocale()`.

Figure 3

Figure 4

```
@DataBoundConstructor
public HelloWorldBuilder
(String name) {
    this.name = name;
}
```

the submitted field.

Also the model must have a getter with the name of the field in the config .xml file to get the data for the second time around when the project is configured again, for example:

```
public String getName() {  
    return name;  
}
```

This information is persisted along with the project configuration, as shown in **Listing 3**. Note that the value of the `name` field is saved as the `HelloWorldBuilder` configuration under the `builders` section.

In the job configuration page, under the **build** section of **HelloBuilder**, if you remove the value from the text field and click elsewhere on the page, a validation error message is displayed (see **Figure 3**).

Because the expression `${app.displayName}` evaluates to `Hudson`, which is the name of the Hudson application, “Hudson” gets inserted into the field help text.

While the UI displays the data of a model, the input from the user in the UI updates the model data when the configuration page is submitted by the user. In this case, the value of the name the user enters in the UI ("winston") is updated in the model.

When the configuration page is submitted, Hudson re-creates the model by passing the corresponding value via the constructor. Hence, the constructor of the model object must have a parameter whose name matches the name of the field. In our configuration we have `<f:entry title="Name" field="name">`.

So the constructor of your `HelloBuilder` must have a parameter named `name`. If you look at the constructor of the class `HelloWorldBuilder`,

```
<?xml version='1.0' encoding='UTF-8'?>
<project>
  <actions/>
  <description></description>
  <keepDependencies>>false</keepDependencies>
  <creationTime>1314407794225</creationTime>
  <properties>
    <watched-dependencies-property/>
  </properties>
  <scm class="hudson.scm.NullSCM"/>
  <advancedAffinityChooser>>false</advancedAffinityChooser>
  <canRoam>>true</canRoam>
  <disabled>>false</disabled>
  <blockBuildWhenDownstreamBuilding>>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>>false</blockBuildWhenUpstreamBuilding>
  <triggers class="vector"/>
  <concurrentBuild>>false</concurrentBuild>
  <cleanWorkspaceRequired>>false</cleanWorkspaceRequired>
  <builders>
    <org.sample.hudson.HelloWorldBuilder>
      <name>Winston</name>
    </org.sample.hudson.HelloWorldBuilder>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>
```

 [See all listings as text](#)

Note: Do not press the Enter (or Return) key. Doing so will submit the configuration page.

If you enter just a two-letter word (for example, **xy**) in the **name** field and click elsewhere, the information message shown in **Figure 4** is displayed.

Let's examine how Hudson displays this information message. Again, as seen in **Listing 3**, `config.jelly` does not include the messages. The magic is in the Jelly file rendering. Some Ajax code is

rendered that, behind the scenes, contacts the Hudson server and asks what message should be displayed.

These Ajax requests are easily observable using Firefox and Firebug. When `config.jelly` is rendered by Hudson, the Jelly tag `<f:textbox />` renders the Ajax code required to do the checking. Firebug displays the Ajax request info shown in **Figure 5**.

The Ajax request shown in **Listing 4** is sent to the Hudson server.



Figure 5



Figure 6

Hudson evaluates this request, finds the extension `HelloWorldBuilder`, executes the method `checkName()`, and returns the result. Again, Hudson uses the convention `doCheck` followed by `{nameOfTheField}` as part of the Ajax URL.

Note: By default, for every `f:textbox` tag in the Jelly config file, Hudson will render the Ajax check. However, if your extension class does not include the corresponding method (in this case, `doCheckName()`), Hudson will silently ignore the check request.

The `doCheckName()` method is straightforward. The Ajax request specifies the `checkName` method with the parameter `value` as `{..}/checkName?value="xy"`. Hence, the parameter `value` of `doCheckName` must be annotated with the annotation `@QueryParameter`. Also, the method must return a `FormValidation` object, which determines the outcome of the check. See **Listing 5**.

The method `doCheckName` returns `FormValidation.error(..)` when an error occurs. The HTML sent back to the client displays the text in red along with an error icon (see **Figure 3**). If `FormValidation.warning()` is returned, the HTML sent back displays the message in yellow along with a warning icon (see **Figure 4**).

Global Configuration

Until now, we have concentrated on how to configure a plug-in at the job level. However, some of the configuration of an extension could be global. For example, if you use Git for source configuration man-

agement (SCM) in a project, you might want to configure two different jobs to force Git to check out from a different repository. So in both the projects, the Git SCM plug-in must be configured to use two different repository URLs.

However, if the Git SCM extension needs to know about the Git native binaries, placing the Git configuration UI that configures the Git binary location at the job level makes little sense, because it doesn't vary from project to project. It makes sense to put such a configuration in Hudson's global configuration page.


For Hudson to include the configuration of a plug-in in its global configuration page, the configuration must be declared in a file called `global.jelly`. The namespace convention for this file is similar to the convention for local configuration. For the `HelloWorldBuilder` global config file, it is `org/sample/hudson/HelloWorldBuilder/global.jelly`.

In Part 1 of this series, we saw that the `HelloWorldBuilder.perform(..)` method includes the code shown in **Listing 6**.

HelloWorldBuilder can be configured to say hello in either French or English. The configuration of which language to use is set globally. Once set, all

LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

```
GET /job/TestProject/descriptorByName/org.sample.hudson.HelloWorldBuilder/
checkName?value=xy HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:6.0.1) Gecko/20100101
Firefox/6.0.1
Accept: text/javascript, text/html, application/xml, text/xml, */*
```

 [See all listings as text](#)

the builds of various jobs that use **HelloWorldBuilder** would say hello in either French or English.

The global configuration of the **HelloWorldBuilder** plug-in is done in the Hudson configuration page in the **Hello World Builder** section (see **Figure 6**).

Here, the user sets the global configuration to control whether French is used as the language. In `global.jelly`, the checkbox is defined as shown in **Listing 7**.

When the user edits the Hudson configuration page, the decision about whether this checkbox should be selected comes from the extension itself. The JEXL expression `_${descriptor}.useFrench()` would resolve to `HelloBuilder.DescriptorImpl.useFrench()`, which is defined as follows:

```
public boolean useFrench() {
    return useFrench;
}
```

`useFrench` is a field in `HelloWorldBuilder`. This field is set to `true` if the user selects

the checkbox. Once the global configuration is submitted, by convention, Hudson calls the method `HelloBuilder.DescriptorImpl.configure()` and passes a JSON object corresponding to the page submission. It is up to the extension to find its submitted value and then apply it. `HelloWorldBuilder` defines this method as shown in **Listing 8**.

In this method, the Boolean value of the field `useFrench` is obtained and used to set `HelloWorldBuilder.useFrench`. The next time `HelloBuilder.perform()` is called during the build of the job, `HelloBuilder.useFrench` is consulted and based on its value, the hello message is written in either French or English.

Conclusion

The Hudson plug-in development environment provides a rich set of extension points with which plug-in developers can develop custom plug-ins. In this article, we explored ways to configure the extensions in a plug-in using the Jelly UI framework . **</article>**

To make the price flash, we create a **Timeline**, which varies the opacity of the cell from 1.0 to 0.1 and back again over a period of one second. The cell is made to flash three times by setting the **cycleCount** appropriately.

Skinning

Another exciting feature of the JavaFX platform is support for cascading style sheets (CSS). Many nodes implement support for CSS, including [TableCell](#).

Adding a style sheet to an application is pretty simple. If you refer back to the `QuoteFXTable` class in **Listing 2**, you will see that after we create the scene in our `Runnable` task, we add a reference to a CSS file to the style sheet list of the scene. This references a CSS file that we include in the same package as the `QuoteFXTable` class.

Now look at the `QuoteFXTableNode` class again in **Listing 3**, specifically the `call` method of the `Callback` class that we pass to the `setCellFactory` method of the price column. Here, we create a new `QuoteFXPriceCell` object and add to the style class list a reference to the name we use in our CSS file to set parameters.

The CSS file is shown below:

```
.fx-table:filled {
  -fx-background-color:
lightblue;
  -fx-border-color: black;
}
```

This is very simple. We create a style that has the name we used in the Java code and specify a pseudo-class that means this style will be used only on price cells that are filled. Two parameters are specified: one to set the color the cell is filled with and one to set the color of the border.

Modifying the style sheet will change the look and feel of the application without the need to change the code. Style sheets are very powerful and can be used to completely change the look of an application when there are numerous JavaFX nodes.

Conclusion

In this article, we've seen how we can migrate a

Swing application to use the richness of JavaFX without having to engage in a complete rewrite. We've modified a sample Swing application to replace its table component in order to make use of the exciting functionality of JavaFX to improve the application's look and feel.

In Part 3, we'll look at ways to replace one of the configuration menus in our sample application. </article>

LEARN MORE

- [JavaFX documentation and tutorials](#)
- [JavaFX 2.0 API documentation \(Javadoc\)](#)
- [JavaFX showcase](#)



YOUR LOCAL JAVA USER GROUP NEEDS YOU

Find your JUG here





Java EE 6 and CDI can be used for lean, efficient implementation of local publish-subscribe communication.

Java EE 6 made the implementation of the Service Activator pattern obsolete. **Listing 4** shows the

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2012

express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. This pattern provides greater network scalability and a more dynamic network topology."

The key point of publish-subscribe is its dynamic nature. Listeners and broadcasters can come and go. There is a dynamic N:M relation between listeners and broadcasters; there is no requirement for the existence of any. A broadcaster could send a message without the existence of any listener. There could be several listeners without any

broadcaster. The “message” in the publish-subscribe definition does not, however, have to be a `javax.jms.Message`. It is better to think about messages as events or payloads. With CDI and EJB 3.1, you can implement lean publish-subscribe communication without JMS.

Java EE 6 (and
CDI, in particu-

lar) comes with a built-in event: the injectable `javax.enterprise.event.Event`. It allows you to fire (distribute) any object you like to all listeners locally. If there is no listener, the broadcasted payload just disappears. To send a message you only have to inject the `Event` class and decide which kind of payload you would

like to distribute. **Listing 6** shows a CDI `String` broadcaster.

The message is sent with a single invocation of the `fire` method. CDI events are transaction-aware and EJB 3.1 beans are transactional by convention. Because there is no configuration (annotation or XML), the broadcast method is going to be executed within a transaction.

The listener is required to implement only a `void` method with a single parameter annotated with the `@Observes` annotation. **Listing 7** shows a CDI `String` listener. The type of the parameter denoted with the `@Observes` annotation must match the type of the fired payload. Only then does the message get delivered; otherwise, it just disappears.


Unlike JMS, with CDI you can easily observe successful and unsuccessful transactions with a single listener at the same time. You only need to set the `during` element of the `@Observes` annotation. **Listing 8** shows an example of how to listen to commits and rollbacks at the same time. The message gets delivered only in the specified transaction phase. The `@Observes(during=)` element makes CDI events particularly useful for the implementation of batch job monitoring or audits. You can easily track all successful and failed transactions.

What About Multiple Destinations?

CDI eventing is type-safe. If the “fired” and “observed” types match, the event gets delivered; otherwise, it is ignored. With the current approach, you can use a type only once. There is no way to distinguish between important and

LISTING 6 / LISTING 7 / LISTING 8 / LISTING 9 / LISTING 10

```
@Stateless
public class MessageBroadcaster {
    @Inject
    Event<String> event;
    public void broadcast(String message){
        event.fire(message);
    }
}
```

 [See all listings as text](#)

tenuous messages. You could, of course, extend the type of the message by wrapping the string with a custom event class. **Listing 9** shows a custom event class for event dispatching.

However, introducing types is a poor approach for mimicking destinations. It results in class explosion and unnecessary bloat. CDI provides an elegant solution with qualifiers. **Listing 10** shows a qualifier for message dispatching. A qualifier is an annotation denoted with the `@Qualifier` interface. `@Retention` should always be set to `RUNTIME`, and with `@Target`, you need to choose the elements to which you would like to apply the annotation. The `@Qualifier` annotation can be considered to be a type extension. For matching, the CDI framework considers not only the types but also the applied qualifiers. You need to use the qualifier only at the injection point:

```
@Inject @Importance  
(Importance.Degree.HIGH)  
Event<String> event;
```

Then, annotate the corresponding parameter:

```
public void onImportantMessage
(@Observes @Importance
(Importance.Degree.HIGH)
String message){}.
```

There is still one deficiency: You need to inject an **Event** for every channel with the given qualifier. Instead of injecting the **Event** with different qualifiers multiple times, you can select the **Qualifier** on the fly, as shown in **Listing 11**, which shows dynamic channel selection. For the on-the-fly selection of the message listener, you only have to pass a **Qualifier** instance to the **select** method. The problem is that it is impossible to instantiate an annotation. **Listing 12** shows a helper class for annotation instantiation.

The class `ImportanceSelector` in **Listing 12** implements the `Importance` annotation and returns its enum value as well as the annotation type. The enum `Degree` is passed as a construc-



The clustering features of GlassFish provide the ability to scale applications horizontally by running a cluster comprising multiple servers.

Note: Two editions of GlassFish exist: GlassFish Server Open Source Edition and Oracle GlassFish Server. This article is relevant to both.

running in a cluster can replicate session data so that a few instances joining and leaving a cluster do not cause applications to become unavailable.

This article is Part 1 of a two-part series on the clustering features of GlassFish and introduces centralized cluster provisioning and management of GlassFish servers. The second article discusses enabling high availability of applications in GlassFish clusters.

Note: The source code for the examples described in this article can be downloaded [here](#).

The running example that we will use in this article is deliberately simple to focus on the management of GlassFish clusters. It is a



Web application called ClockEE that provides a single page on which the current time is displayed (see **Figure 1**).

The application is a [Maven](#) project whose source code is in the ClockEE.zip file. The sole view is provided using JavaServer Faces, as shown in **Listing 1**.

In turn, it requires the presence of a Contexts and Dependency Injection (CDI)-managed bean that provides the clock value, as shown in **Listing 2**.

The @Named annotation



Julien Ponge provides a brief introduction to his GlassFish series.

PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

JAVA IN ACTION

JAVA TECH

ABOUT US

○

f

ava
net

log



55

```
$ asadmin list-domains  
domain1 running  
Command list-domains executed successfully.  
$ asadmin create-cluster my-first-cluster  
Command create-cluster executed successfully.  
$ asadmin list-clusters  
my-first-cluster not running  
Command list-clusters executed successfully.
```

By creating a local instance, GlassFish automatically created a local *config node*. (We will later explore the options available when creating instances for remote systems.)

```
$ asadmin deploy --target my-first-cluster
```

A **REST Web service interface** is available for easily integrating GlassFish with third-party management systems. There is also a Web administration console that provides an intuitive graphical interface.

is propagated to the node instances when synchronization occurs.

Two distinct types of nodes exist:

- *Config nodes* need to be locally provisioned, configured, and administered. They are declared to a cluster from the DAS, but the DAS cannot perform any administrative commands. Administrators are required to log in to the machines running such nodes and perform the required tasks. Such nodes cannot be centrally administered from the DAS.
- *SSH nodes* can be completely managed from the DAS. The minimal requirement is to have an SSH server and a Java SE runtime environment on such nodes. Another benefit of SSH nodes is that communications between the DAS and the nodes is secure, both from an authentication

NODE NOTE

SSH nodes can be **completely managed** from the DAS.

and a confidentiality point of view.

SSH nodes are preferable, whenever possible, because the complete management of a GlassFish cluster can be performed from the DAS host machine.

Given that SSH nodes arguably provide more-interesting features, we will focus solely on them in the remainder of this article. Adding Config nodes to a cluster requires preparing the node hosts by installing GlassFish, creating local Config nodes on each of the hosts, and declaring the nodes to the DAS. As the next section shows, these tasks can be fully automated in the case of SSH nodes.

Remotely Provisioning and Adding an SSH Node

In this section, we are going to see how to remotely provision and add a

GlassFish instance to our cluster. It is assumed that you have a valid user account on a remote server that has, at the very least, a working Java SE 6 runtime environment and an OpenSSH server.

For the purpose of testing, you could do this on your local machine as well, if OpenSSH

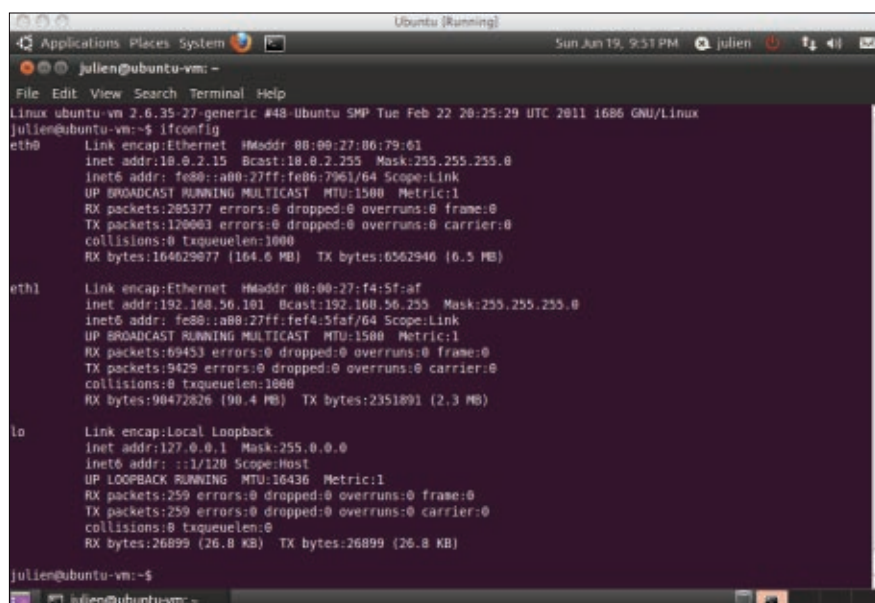


Figure 4

LISTING 9

LISTING 10

LISTING 11

LISTING 12

```
$ asadmin setup-ssh --sshuser
julien 192.168.56.101
Enter SSH password for julien@192.168.56.101>
Copied keyfile /Users/julien/.ssh/id_rsa.pub to julien@192.168.56.101
Successfully connected to julien@192.168.56.101 using keyfile
/Users/julien/.ssh/id_rsa
Command setup-ssh executed successfully.
```



[See all listings as text](#)

is running. A solid alternative that I experimented with while writing this article is to use a virtual machine. I used [Oracle VM VirtualBox](#) to virtualize a Linux Ubuntu Maverick operating system (see **Figure 4**).

Given that operations happen over SSH, any action that requires connecting to the remote servers will require authentication. By default you will have to type your password every time, but you can drastically simplify this. Password-less SSH connections are possible by generating a public/private key pair locally and adding the public key fingerprint to the `~/.ssh/authorized_keys` file on the remote servers.

Although you can do this manually, GlassFish can automate it, as shown in **Listing 9**.

Let's assume that the remote server does not have GlassFish. Let's install

it on the remote machine in `/home/julien/glassfishv3` by performing a remote provisioning. See **Listing 10**.

A working GlassFish application has just been provisioned remotely from a local image that was copied over SSH. We can now create an SSH node on the remote server:

```
$ asadmin create-node-ssh  
--nodehost 192.168.56.101  
--installdir  
/home/julien/glassfishv3 ubuntuvm  
Command create-node-ssh executed  
successfully.
```

As shown in **Listing 11**, we can create an instance on this SSH node, name it `vm1`, and then add it to `my-first-cluster`.

By default, the newly created instance is not running, so we can start it. See **Listing 12**.

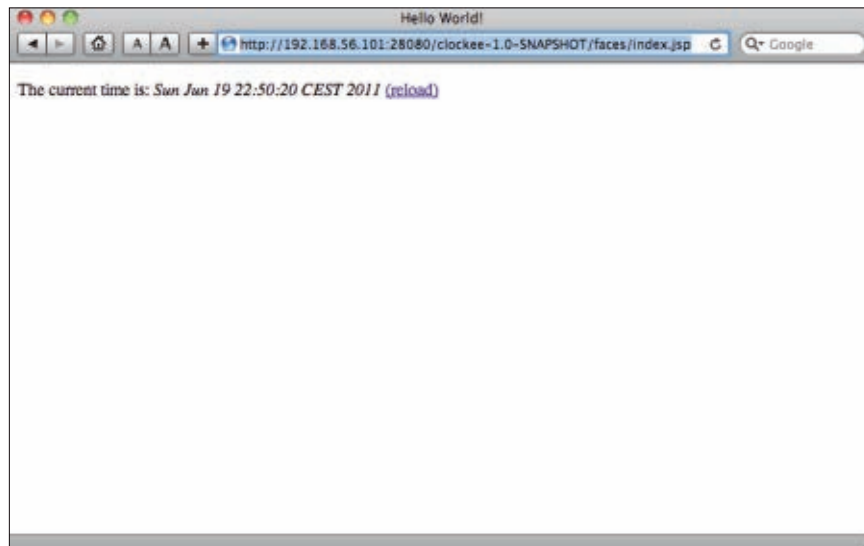


Figure 5

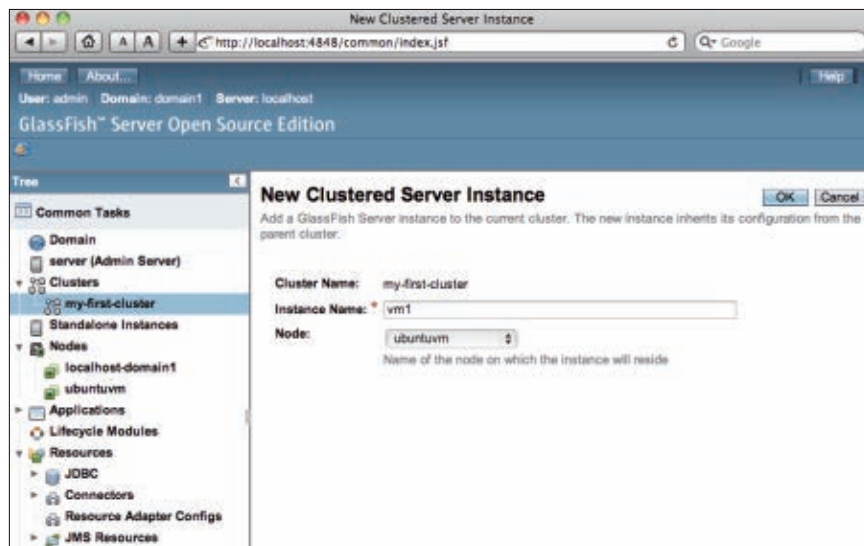


Figure 6

We can check that ClockEE was transparently deployed to the `vm1` instance when it was launched, provided that it had already been deployed to `my-first-cluster` (see **Figure 5**).

How About the Web Administration Console?

We did all administration work from the command-line using `asadmin`, but

as was mentioned earlier, that is not the only option. A REST Web service interface is available for easily integrating GlassFish with third-party management systems. There is also a Web administration console that provides an intuitive graphical interface.

Both are on par with the equivalent **asadmin** shell commands.

Figure 6, Figure 7, and Figure 8

provide a few screenshots of using the Web administration console to perform a few tasks we did from the command-line before.

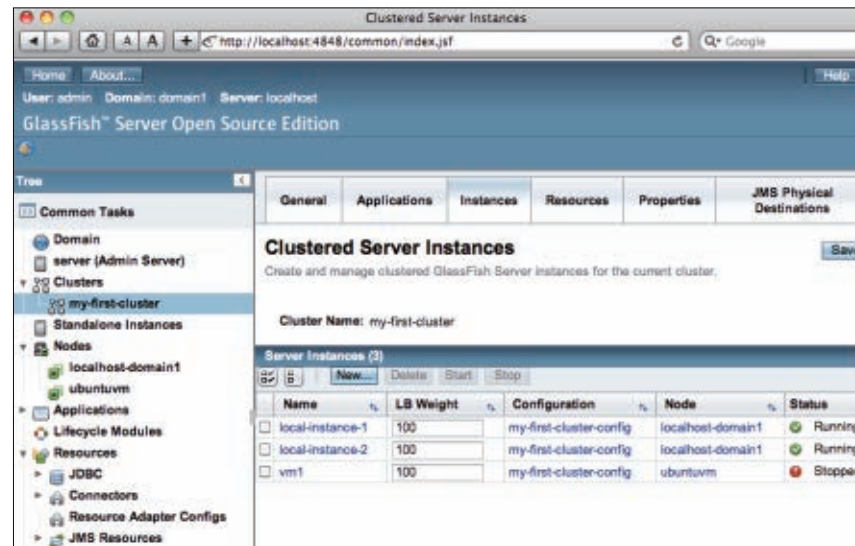


Figure 7

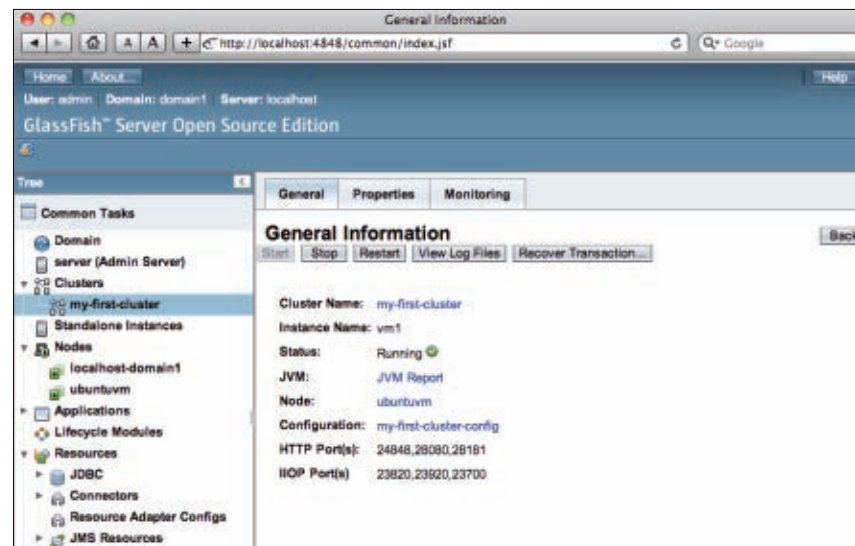


Figure 8

Conclusion

This article introduced the centralized cluster provisioning and management of GlassFish clusters. The ability to get a new host up and running remotely as long as it has an SSH server and a Java SE runtime is especially powerful. We deployed a simple stateless application to a cluster and played with common administration commands to check the state of our cluster and manipulate instances.

The next article in this two-part series shows how to enable high availability of stateful applications deployed to our cluster by providing transparent session failover. [.</article>](#)

LEARN MORE

- [GlassFish product documentation](#)
- ["Clustering in GlassFish Version 3.1"](#)
- [GlassFish videos on YouTube](#)
- [*Real World Java EE Night Hacks—Dissecting the Business Tier* by Adam Bien](#)
(press.adam-bien.com, 2011)
- [*Beginning Java EE 6 with GlassFish 3* by Antonio Goncalves](#) (Apress, 2010)
- [*The Java EE 6 Tutorial: Basic Concepts*](#)
fourth edition by Eric Jendrock,
Ian Evans, Devika Gollapudi, Kim Haase,
and Chinmayee Srivathsa
(Prentice Hall, 2010)

STAY TUNED

The next article shows how to enable high availability of stateful applications on the cluster.



Oracle Technology Network: Your Java Nation.

Come to the best place to collaborate with other professionals on everything Java.

Oracle Technology Network is the world's largest community of developers, administrators, and architects using Java and other industry-standard technologies with Oracle products. Sign up for a free membership and you'll have access to:

- Discussion forums and hands-on labs
- Free downloadable software and sample code
- Product documentation
- Member-contributed content

Take advantage of our global network of knowledge.

JOIN TODAY ► Go to: oracle.com/technetwork/java

ORACLE®



VINICIUS SENER AND
YARA SENER



Wake Up and Smell the Coffee

The creators of jHome, a 2011 Duke's Choice Award winner, demonstrate why Java EE 6 development is so easy.

jHome is a complete home-automation open source API based on GlassFish Server Open Source Edition and Java EE 6, which enables developers to control anything in their homes.

In this article, we describe the jHome project in detail, including the technical context, code, and hardware details. Usage examples show the power and ease of use of Java EE.

Like most Java programmers, we are crazy for coffee. We always wanted to schedule our coffee maker to turn on at a certain time, so we could wake up to the smell of coffee. So we set out to develop a way to do this. After all, nothing is more fun than using the Java EE 6 Timer Service to schedule the “real objects” of your life.

People who always lose or forget their keys, for instance, might want to control their front door through a Website or a mobile application. And why not open your door by sending a message to a Twitter profile that is associated with your jHome application? Others might want to control their balcony lights or swimming pool

lights from their mobile devices. The possibilities for simplifying the way you control the objects in your life are unlimited.

jHome Overview

jHome is a set of interfaces and components based on Java EE 6 for implementing home-automation solutions. Using Enterprise JavaBeans (EJB), the Timer Service, Java API for RESTful Web Services (JAX-RS), Contexts and Dependency Injection (CDI), and Web components, jHome lets you automate your house, build a Web or mobile app, and control things over the internet.

Figure 1 shows an overview of the jHome architecture.

With the simple code shown in **Listing 1**, you can inject an object that represents a lamp in your house and turn it on or off using a Servlet.

Using EJB beans to control a lamp may sound strange, but the Singleton, Transactions, and Injections resources are a very useful way to deal with hardware communication, concurrency, and queuing requests.

jHome Devices

jHome is based on open source hardware and does not require proprietary or expensive solutions. The jHome team created the reference implementation hardware based on [Arduino](#), an open source electronics prototyping platform. The cost to control two lamps or wall sockets is less than US\$100 using Arduino and USB cables.

Each jHome device can communicate with the jHome application deployed on the Java EE container using different protocols and different wired and wireless components:

- USB cable (the cheapest solution)
- Bluetooth (good for wireless prototypes)
- ZigBee/xBee (the best wireless solution for real implementations)

Relay boards are very useful because they let you control wall sockets, lamps, and any other electronic device. jHome can also read values from sensors such as temperature, light, distance, and gas.

The standard firmware for jHome devices can control relays, dimmers, RGB LEDs, motors, and robots, and code for various sensors is available at the Tools Cloud

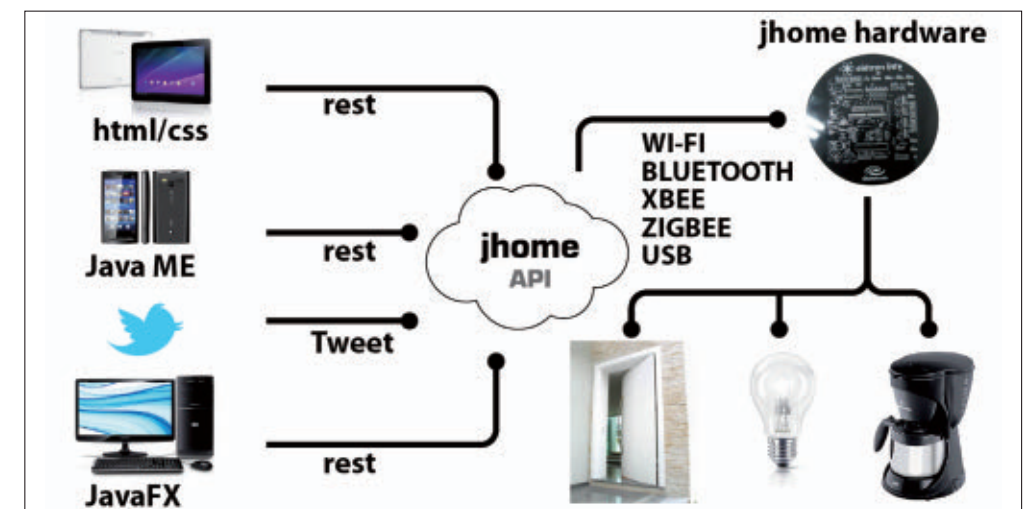


Figure 1

GIT repository. Use username jhome and password: jhome. Let's clone!

Another interesting implementation we made was to integrate jHome with a heart rate transmitter and program jHome to do a certain task (such as open a Website or send a message) when the user achieves a predefined maximum or minimum heart rate. In the Java EE world this would be a Heart-Driven Bean—something that brings the fun back to development.

Understanding Arduino and Open Source Hardware

Arduino is an open source electronics prototyping platform based on flexible, easy-to-use hardware (a printed circuit board) and software. There are several different implementations varying in size, price, and memory capacity.

The programming language used to program the board is a C-like language, and there is a Java IDE that helps you to create programs and send them to the board. Although the language running on the Arduino board is not Java, the syntax is easy for experienced Java programmers to learn and understand.

Java EE 6 and jHome

Java EE has really evolved over the last 10 years. It's still powerful but is now much easier to use, affordable, and lightweight—in fact, our cell phones can now run a Java EE container, such as GlassFish Server Open Source Edition.

Controlling your house with a Java EE application server is now a reality. All Java EE components are useful for auto-

maintaining control of your house. Here's how the jHome team used some of the components in the jHome implementation.

EJB: core components. EJB beans are used for all jHome core components. The EJB resources used in this application are the following:

- Singleton: Used to represent devices and boards that cannot have more than one instance
- Transaction: Used to deal with hardware concurrency
- Web services: Very useful to expose functionality to any kind of client application
- Timer Service: Used to schedule wall sockets, update sensor values, check for Twitter messages, and so on

Timer Service: scheduling the coffee maker. jHome uses the Timer Service API to allow you to schedule services such as turning lights on and off and scheduling the coffee maker.

And because coffee is a very critical resource for Java developers, it's important to have high availability. The Timer Service API delivers this: if you restart your application server, you will not need to reschedule the coffee maker.

Servlet: controlling lights via the internet. There are a couple of Servlets for controlling lights. There are LED strips that can control color using RGB. You can control the color of your swimming pool lights, aquarium lights, or any other light.

jHome Architecture

jHome is modular and uses [Maven](#). Let's look at its modules.

LISTING 1

LISTING 2

```
@WebServlet(name = "Light", urlPatterns = {"/Light"})
public class Light extends HttpServlet {
    private static boolean on;
```

```
@EJB
Relay light;
```

```
@Override
protected void doGet(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException {
    if (on) {
        light.turnOff("lamp");
        on = false;
    } else {
        light.turnOn("lamp");
        on = true;
    }
}
```



[See all listings as text](#)

jHome architecture. jHome has three main modules:

- **jHome Interface** : All jHome interfaces can be found in this module.
- **jHome Core**: This module has all the interface's implementation and is responsible for discovering the jHome devices available in the house using an open protocol, reading their sensors, and also helping to expose the jHome device functionality to other Java components.
- **jHome Web**: A Web application that uses HTML5 and jQuery to control the default functionalities from jHome: turn a lamp on and off, control the RGB color LEDs, schedule a coffee

maker, and read basic sensors.

You must deploy the jHome Interface and Core modules to run the basic jHome services. The jHome Web module is optional, and you can create or use any other client applications such as Java ME, JavaFX, or Gingga-J to run on your TV.

Sensor API. The Sensor API provides information about the physical environment, and it is implemented using the Timer Service.

With the jHome Sensor API you can feed your Java application or Website with information from different sensors that are plugged into your jHome devices. This capability provides many new possibilities for designing your

JAVA IN ACTION

- # JAVA TECH

ABOUT US



blog



62

- jHome
- Arduino

Computers, Printers, Routers, BlackBerry Smartphones, Cell Phones, Kindle E-Readers, Parking Meters, Vehicle Diagnostic Systems, On-Board Computer Systems, Smart Grid Meters, Lottery Systems, Airplane Systems, ATMs, Government IDs, Public Transportation Passes, Credit Cards, VoIP Phones, Livescribe Smartpens, MRIs, CT Scanners, Robots, Home Security Systems, TVs, Cable Boxes, PlayStation Consoles, Blu-ray Disc Players...



#1 Development Platform

ORACLE®

oracle.com/goto/java
or call 1.800.ORACLE.1



Using the Location API, you can access location data and incorporate that data into a game.

Note: The source code for the game I develop in this two-part series can be downloaded [here](#).

Using the Location API

Listing 1 shows the initialization code for the Location API

Our MIDlet should implement the `LocationListener` interface, because if the provider is found, the MIDlet can listen to the events that the provider generates. The most important event is, of course, updates on current coordinates. The `locationUpdated` method implementation handles this in our code, as shown in **Listing 2**.

The next step is to find players near your own location with whom you can play the game, as shown in **Listing 3**. The `getPlayersFromServer` placeholder mock method returns a list of





Figure 2

players. The `findPlayers` command is handled by the `commandAction` method.

For the best results, make sure any code that might take a long time to return, such as querying the server for a list of players, is handled in a separate thread so it doesn't block the main program flow. The end user should be able to cancel the query from the server if required, and the program should handle that implicitly. **Figure 2** displays a list of players.

Handling the OK button for the selection of player launches the game, as shown in **Listing 4**.

The `initiateGame` mock notifies the server and then launches the game using the `launchGame` method, as shown in **Listing 5**.

For trivial applications, this should suffice, but for production applications, there are other aspects that would require attention as well, such as better error handling, increased sensitivity of detection, valid response from the server, and so on.

LISTING 1

LISTING 2

LISTING 3

LISTING 4

LISTING 5

```
try {

    // create very basic criteria for the location provider
    Criteria criteria = new Criteria();
    criteria.setHorizontalAccuracy(Criteria.NO_REQUIREMENT);
    criteria.setVerticalAccuracy(Criteria.NO_REQUIREMENT);
    criteria.setPreferredResponseTime(Criteria.NO_REQUIREMENT);
    criteria.setCostAllowed(false);

    // get the provider based on the criteria
    LocationProvider provider = LocationProvider.getInstance(criteria);

    // provider found
    if (provider != null) {

        // set this MIDlet to be the listener for location changes
        provider.setLocationListener(this, -1, 0, 0);

    } else {
        display.setCurrent(
            new Alert("Error!",
                "Phone does not support Location Provider with the given criteria",
                null, AlertType.ERROR));
    }

} catch (Exception ex) {
    handleError(ex);
}
```



[See all listings as text](#)

Conclusion

In this second article, we discussed how to incorporate the Location API into the gaming code that we developed in the previous article. We also saw that our core MIDlet needs to implement the `LocationListener` interface and handle the `LocationUpdated` method (at the very least). In addition, we discussed how to form very basic criteria for

accessing location data and we saw how to incorporate the data we get into our game. [</article>](#)

LEARN MORE

- Tutorial: ["Using the Location API for Favorite Spots"](#)
- Tutorial: ["Working with the Mobile Sensor API"](#)



GIVE BACK! ADOPT A JSR

[Find your JSR here](#)



The stereotype of the lone-wolf programmer locked away in a room has faded in recent years. Team software development is the norm, and pair programming is an accepted Agile practice. Communities with different strengths grow up around a language. With such considerations in mind, developers might want to set up an environment where they can test-drive several new languages to quickly compare them. Java—specifically the Java Virtual Machine (JVM)—provides an excellent platform for such an exercise.

The Java language is not—and was never intended to be—the perfect programming language for every situation. The JVM is software constructed to execute bytecode compiled from Java. Each JVM implementation is designed to run on a specific platform. The original vision was to allow Java to be compiled once and run on a variety of platforms. But the JVM has also been leveraged in recent years to run bytecode created by other programming languages.

RUN AND COMPARE

The Apache Maven

project presented here can be set up in minutes and allows developers to run and compare scripts written in five dynamic scripting languages: Clojure, Groovy, JavaScript, JRuby, and Jython.

Start Your Engines

Trying out Clojure, Groovy, JavaScript, JRuby, and Jython requires an initial setup in order to find, download, install, and configure each implementation. The project described here reduces the time and effort required by using Maven to locate and download resources from various public code repositories. Maven is also used to build the project so

that a single executable JAR is created, which contains a class that can be used to run scripts written in several different languages.

Setup

To run the project, do the following:

1. Validate your Java installation by running the following commands:

```
javac -fullversion
java -fullversion
```

The output from each of these commands should indicate a Java SDK version of 1.6. If it does not, download and install a Java SDK (the Microsoft Windows version is `jdk1.6.0_14`).

2. Download and install [Maven](#), and make sure it is included in your [PATH](#) environment variable. Then run the following command:

- mvn -version

The output of this command should indicate that a recent version of Maven is installed (for example, version 3.0.3).

3. Download the project itself, which consists of a small amount of Java code, a few scripts, and a pom.xml file. Extract the code, and navigate to the directory that contains pom.xml.
4. Build the code, which will download JAR files from various archives and assemble the application:

- mvn clean install

All downloaded JAR files are installed in your local Maven repository (by default, in your home directory under `.m2/repository`).

5. Run the application from within Maven (this works regardless of the platform in use). To see the list of available scripting languages, run the following command:

- `mvn -q exec:java`

Note: You can also run the file by using the Java interpreter and referencing the JAR itself. The `jvms.bat` script contains such a call for a Microsoft Windows environment, and the `jvms.sh` script contains such a call for a Mac OS environment. The remainder of this article uses “`mvn`” to avoid operating-system-specific details.

6. To compare various languages, pass at the command line the scripts that are to be evaluated.

One or more files can be passed as arguments, and the files can use the same or different scripting languages. The scripts used in this article reside in `src\main\resources\scripts`. The customary “Hello World” example can be run as follows:

```
mvn -q exec:java -Dexec.args="hello.clj hello.js hello.groovy  
hello.rb hello.py"
```

Additional Scripts

In addition to the “hello” scripts, several other scripts (described in a bit more detail below) are provided that can get you started with your own experiments.

You can run these scripts and modify them to compare and contrast Clojure, Groovy, JavaScript, Python, and Ruby. In some cases, the code is nearly identical, but in other cases, it is radically different. When you run a set of scripts, you will also notice that the output shows a time in nanoseconds for each script. This allows you to create scripts in several



In the last issue, Arun Gupta gave readers a piece of code and asked if the code is guaranteed to work in a multithreaded environment.

The correct answer is #2: No. Servlets are not re-entrant, and EntityManager is not threadsafe.

Instead inject as

```
@PersistenceUnit EntityManagerFactory em;
```

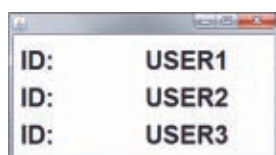
and then get EntityManager within the doGet() method.

This issue's challenge comes from Marek Piechut, a technical leader at Transition Technologies S.A. in Łódź, Poland, who is currently working on a heavy Swing client for a large enterprise application.

1 THE PROBLEM

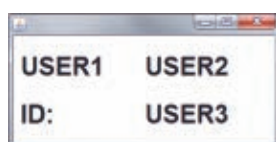
Java Swing is the main Java framework for heavy client implementations.

Let's create a Swing window that displays a simple table. What we want displayed is



ID:	USER1
ID:	USER2
ID:	USER3

But what we get is



USER1	USER2
ID:	USER3

ART BY I-HUA CHEN

2 THE CODE

Consider the following code when preparing a window:

```
public class MyWindow extends JFrame {
    public MyWindow() {
        GridLayout layout = new GridLayout(0, 2, 4, 4);
        setLayout(layout);
        JLabel idLabel = new JLabel("ID:");

        for (String d : new String[]{"USER1", "USER2", "USER3"}) {
            add(idLabel);
            add(new JLabel(d));
        }
    }
}
```

3 WHAT'S THE FIX?

- 1) The first value for GridLayout is invalid and should be replaced with the actual number of rows.
- 2) You can't add elements directly to JFrame; you should first pack all labels into JPanel and add it to JFrame.
- 3) You should replace a single shared instance of idLabel with a new JLabel instance for each row.
- 4) You need to call layout.layoutContainer(this) in the constructor end to lay out the window properly.



→ GOT THE ANSWER? Look for the answer in the next issue. Or submit your own code challenge! ←

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2012